TURING

引言：软件和编程艺术的兴起

FORTRAN：早期的“转折点”

20世纪60年代的惨痛教训：从繁盛到COBOL和IBM 360计划成为现实

打破巨型计算机的控制：Unix和C

为大众编程：从达特茅斯的BASIC到Visual Basic

欧洲的影响力：从Algol到Pascal再到C++

属于自己的计算机：PC产业的起步及Word的故事

服务于大众的计算机：从Gooey到Macintosh的漫漫长路

为每一个人编程：让用户自己动手

Java：杂乱中诞生的新语言

一定有更好的方式：Apache和开源运动

Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists and Iconoclasts — the Programmers Who Created the Software Revolution

软件故事

谁发明了那些经典的编程语言

[美] 史蒂夫·洛尔◎著　　张沛玄◎译

聆听软件行业发展的精彩故事　领悟软件巨擘的深邃思想　放飞想象力，通过编码改变世界

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

TURING

Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists and Iconoclasts — the Programmers Who Created the Software Revolution

软件故事

谁发明了那些经典的编程语言

[美] 史蒂夫 · 洛尔◎著

张沛玄◎译

人民邮电出版社

北京

图书在版编目（C I P）数据

软件故事 ： 谁发明了那些经典的编程语言 / （美）
洛尔（Lohr,S.）著 ； 张沛玄译. -- 北京 ： 人民邮电出
版社，2014.7
ISBN 978-7-115-35508-9

Ⅰ. ①软… Ⅱ. ①洛… ②张… Ⅲ. ①软件－基本知
识 Ⅳ. ①TP31

中国版本图书馆CIP数据核字(2014)第097634号

内 容 提 要

本书介绍了多种语言和软件的起源以及促进软件行业发展的重大成就，以传记体讲述了埋没于历史洪流却起到了关键作用的编程人员及其贡献，包括“存储式计算”早期出现的女性软件先驱的故事。本书内容主要包括：约翰·巴克斯发明 Fortran 语言、约翰·麦卡锡设计 Lisp 语言、“COBOL 之母”葛丽丝·霍普等人创建 COBOL 语言、肯·汤普森与丹尼斯·里奇开发 Unix 操作系统和 C 语言、托马斯·库尔兹与约翰·凯默尼开发 BASIC 语言、本贾尼·斯特劳斯特卢普开发 C++、“Word 之父”查尔斯·西蒙尼研发 Word、阿兰·凯伊设计 Smalltalk 语言、安迪·赫兹菲尔德等研发 Macintosh、钱柏林等创建 SQL 语言、詹姆斯·高斯林发明 Java，等等。

本书适合计算机相关从业人员及对软件行业感兴趣的读者参考阅读。

◆ 著　　[美] 史蒂夫·洛尔
译　　张沛玄
责任编辑　李松峰　毛倩倩　李　瑛
执行编辑　姜力心　杨　琳　董瑞霞
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：720×960　1/16
印张：17
字数：244千字　　　　2014年 7 月第 1 版
印数：1 – 4 000册　　　　2014年 7 月北京第 1 次印刷

著作权合同登记号　图字：01-2012-2442号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版 权 声 明

译　者　序

要想真正把事情搞得一塌糊涂，还得依靠计算机。

这是我真正见到计算机之前看到的一句话。

后来，我学习了 BASIC、FORTRAN、C 和其他一些语言，也写过一些大大小小的程序。我曾为调试程序苦思冥想，也曾为自己的“作品”沾沾自喜，却从未认真想过这些编写程序的程序从何而来。那些创造出如此美妙东西的，该是怎样神奇的人？

所以，当我看到这本介绍软件发展史的书时，它立刻就引起了我的兴趣。

本书不但详细介绍了计算机各个发展阶段中主流软件的产生过程，而且讲述了很多鲜为人知的故事。特别值得一提的是，书中讲述了众多人物的故事，其中不乏一些大名鼎鼎的业界传奇人物，当然也有不少不为人知的“无名小卒”。但他们都是创造了软件历史的人，都是值得我们尊重的人。

作者史蒂夫·洛尔曾是《纽约时报》的技术、商业和经济专栏作家。为撰写本书，他进行了大量调查、研究和走访。本书英文版于 2001 年由 Basic Books 首次出版。

由于译者水平有限，书中难免有错误和疏漏之处，欢迎读者批评指正。我的邮箱：cx10@sina.com.cn。

致　谢

1999 年春天，吉姆 · 格雷[①]去领取图灵奖（计算机科学领域的诺贝尔奖）时途经纽约，让我萌生了撰写本书的想法。格雷是土生土长的加利福尼亚人，20 世纪 60 年代就读于加州大学伯克利分校。毕业后，他大部分时间都在硅谷工作。他对硅谷的商业周期了如指掌，对当时盛行的电梯游说和 IPO 等互联网投资热潮不以为然。当时，人们似乎更加热衷于赚钱而非关注技术——按照格雷的标准来说，这是个扭曲的世界。

他说，诚然，这个领域中有才能的人生活得更好。“但这与钱无关，”他指出，“真正的乐趣和吸引力在于能够运用这种超酷的软件技术应对和创造事物。”

① Jim Gray（1944—2007），著名的数据库专家、事务处理技术权威，是 1998 年图灵奖得主。2007 年 1 月 28 日独自驾船出海而失踪。——编者注

我认为，对计算机编程的历史进行深入探索，并与构建和缔造软件世界的大师进行对话，也许是一件非常有趣的事情。计算机是相对新兴的行业，而且发展十分迅速，所以大部分编程先驱都还健在，这又是一个便利之处。

因此，当时看起来撰写本书是个不错的想法。

有些人也认同这是一项有趣的工程，在此我要向他们表示感谢。感谢Brockman公司的约翰·布罗克曼和卡廷卡·麦特森，要不是他们的鼓励，我可能不会开始创作本书。感谢Basic Books出版社决定出版本书，编辑主任伊丽莎白·马奎尔和编辑威廉·莫里森都思维缜密、敏捷干练。感谢《纽约时报》，尤其是执行编辑约瑟夫·莱利维尔德，他慷慨地批准我休假，而我休假的时间超过了预期。

感谢哥伦比亚大学新闻研究生学院院长汤姆·戈尔茨坦，在我离开《纽约时报》期间他热情地为我提供工作场所。

向艾尔弗雷德·斯隆基金会以及负责公共科技常识项目的主管多伦·韦伯致以最诚挚的感谢，因为得到帮助我才得以完成本书。

感谢那些专业的教育机构大力支持我开展研究。美国计算机协会允许我访问他们的数字图书馆，这是难能可贵的。查尔斯·巴贝奇研究所的口述历史项目也是我重要的资料来源。计算机历史中心博物馆（2000年更名为计算机历史博物馆）资助的录像讲座对我也非常有帮助。

特别要感谢那些接受我采访的人，他们付出了大量的时间，展现了无穷的耐心与智慧。他们经常受到我的打扰，要接受当面采访、接听电话或回复邮件，采访冗长而又烦琐，有时还要多次重复，有的人甚至还要审阅部分手稿。他们包括亚历克斯·艾肯、弗兰·艾伦、丹尼斯·艾利森、马克·安德列森、约翰·巴克斯、珍·巴蒂克、布莱恩·贝伦多夫、罗伯特·贝莫、蒂姆·伯纳斯–李、丹·布里克林、小弗雷德里克·P. 布鲁克斯、汤姆·巴顿、马丁·坎贝尔–凯利、彼得·卡佩克、史蒂夫·卡普斯、唐·钱伯林、阿兰·库珀、乔治·康乐瑞斯、理查德·道金斯、道格·恩格尔巴特、鲍勃·弗兰克斯顿、

约翰·盖奇、比尔·盖茨、理查德·戈德堡、詹姆斯·高斯林、吉姆·格雷、萨提什·顾普塔、洛伊丝·海波特、安迪·赫兹菲尔德、安德斯·海尔斯伯格、托尼·霍尔和瓦茨·汉弗雷等。

还要感谢比尔·乔伊、菲利普·卡恩、霍华德·卡兹、阿兰·凯伊、肯·肯尼迪、布莱恩·柯尼汉、高德纳、托马斯·库尔茨、J. A. N. 李、巴特勒·兰普森、约翰·麦卡锡、帕米拉·麦可杜克、丹·麦克拉肯、道格拉斯·麦克罗伊、罗杰·尼达姆、约翰·H. 帕尔默、拉吉·瑞迪、丹尼斯·里奇、珍·萨梅特、戴维·赛尔、埃里克·施密特、卡尔·夏皮罗、弗雷德·夏皮罗、迈克·谢里丹、约翰·肖奇、查尔斯·西蒙尼、理查德·斯托曼、盖伊·斯蒂尔、本贾尼·斯特劳斯特卢普、兰迪·特布什、查尔斯·泰克、肯·汤普森、林纳斯·托瓦兹、约瑟夫·特劳布、盖伊·特里布尔、亚瑟·范·霍夫、莫里斯·威尔克斯、王文科、理查德·索尔·沃尔曼以及欧文·齐勒等。

此外，还要感谢我的朋友和同事约翰·马尔科夫。感谢他的忠告以及在旧金山容我借宿，也感谢他的耐心，在我销假回去工作后，他才请假写作他的书。另外，还要谢谢他的妻子莱斯利·特齐安热情款待我这样一位常客。

最后，一如既往地感谢特里和尼基。

目　　录

第 1 章

引言：软件和编程艺术的兴起

秋高气爽的黄昏，凭窗远眺，华盛顿湖碧波荡漾，一叶孤舟划过，泛起层层涟漪。远处的地平线上，依稀可见西雅图市的轮廓。这就是从查尔斯·西蒙尼位于华盛顿湖畔的家中看到的景象。1966 年，这位年仅 17 岁的电脑编程奇才离开了匈牙利首都布达佩斯，远走他乡，从此开始了非凡的人生历程。他这幢房子依山傍水，占地 1800 多平方米，配有图书馆、计算机实验室、健身中心和游泳池。不过，从公路上几乎完全看不到。整套房子从内到外都由玻璃、木头和钢材建造而成，堪称现代主义的典范。黑色的磨光石地板闪闪发光，访客需要换鞋方可进入。墙上仅挂了几幅洛伊·李奇登斯坦、贾斯培·琼斯和维克托·瓦萨雷里的现代主义画作。除了这些艺术品，西蒙尼还收藏喷气式飞机。

他收藏了两架飞机，其中包括他驾驶过的一架已经退役的北约战斗机。此外，他慷慨大方，曾捐赠数百万美元支持高校科研，牛津大学因此设立了一个以他的名字命名的教授席位，普林斯顿高级研究院还将他的名字刻在了数学系大楼上。年少的西蒙尼逃离匈牙利时一无所有，但是现在，已是亿万富翁的他却视金钱为无物。他说："我现在已经不再为了逐利而做事。"[1]

谈起他所取得的成绩，西蒙尼把这一切都归功于软件业的兴起，以及他在编写计算机代码方面的天赋异禀，当然，还有机遇、运气以及资本市场的反复无常。他的职业生涯始于20世纪60年代中期，那时他在匈牙利中央统计局工作，有点像美国青少年电脑黑客。他涉猎广泛，不断充实自己，自学了如何在俄制的 Ural II 电脑上进行编程。论计算机发展水平，布达佩斯还处于20世纪50年代早期的技术水平，比西方落后了几十年。多年来，软件不断发展，编程人员的视线不再局限于二进制数字或数位——机器语言 0 和 1。但是，西蒙尼尝试用纯粹的机器语言与计算机对话。"那是编程的石器时代，"他回忆道，"我穿越了时间隧道。"

移民美国后，西蒙尼就把他的名字卡罗利改为了查尔斯。他先后进入加州大学伯克利分校和斯坦福大学学习，毕业后就职于施乐公司帕洛阿尔托研究中心（Xerox PARC）。那时正值20世纪70年代的黄金时期，他们的团队进行了大量的研究和开发工作，正是这些工作使人们使用个人电脑的方式初具雏形。在施乐帕洛阿尔托研究中心，西蒙尼是 Bravo 最重要的开发者。Bravo 是一种全新的书写和文本编辑程序，可以在电脑屏幕上显示单词，就像熟练的排字工捡字那样。这就是后来广为人知的"所见即所得"（WYSIWYG，What You See Is What You Get）功能，它开启了桌面出版（desktop publishing）的大门，个人电脑从而成为提高个体创造力的工具。

但是，施乐公司并没有真正意识到帕洛阿尔托实验室工作的重要性。西蒙尼看透了这一点，并开始另谋高就。1980年夏季，他低调地拜访了西雅图城外一家在个人电脑行业初出茅庐的小公司——微软。虽然公司刚刚起步，仅有 40

名员工，但在这里，西蒙尼看到了希望。他和比尔·盖茨一见如故，于是加入了微软。

微软的 Word 文本编辑器如今已是世界上使用最广泛的软件程序之一，它是 Bravo 的商业化产物，西蒙尼则是当之无愧的“Word 之父”。在他看来，个人电脑只是软件的传递载体，给用户很大帮助，并使编程人员如虎添翼。“写下几行代码，瞬间就能改变亿万人的生活，”他说，“这就是软件。”[2]

从微软退休前的最后几年中，西蒙尼一直致力于一个重要的研发项目，旨在大大提高计算机编程人员的生产力。他认为编程人员目前使用的工具和方法依然过于原始，限制了人类智力在软件开发过程中的发挥，从而阻碍了软件业的发展。尽管如此，西蒙尼还是对软件的兴起赞叹不已：“这表明软件的力量强大无比。即使用现有的简单工具，软件也能做那么多事，这太神奇了！”

战后，软件业作为一个探索领域、一个行业以及通信与商业之间的媒介，取得了令人瞩目的迅速发展，而这一切几乎都发生得悄无声息。计算机编程的起源至少可追溯到 19 世纪。当时英国数学家查尔斯·巴贝奇正致力于解决分析机的计算问题；分析机是现代计算机概念化的雏形。今天，我们可以将其所做的事情称为编程。编程中最基本的概念是算法，即执行某种操作的一组指令，或者说是计算的方法。追根溯源的话，算法（algorithm）最早出现在巴比伦[3]，而 algorithm 这个词则是对波斯学者 Muhammad ibn Musa al-Khwarizmi[4]姓氏的讹传，他写过一篇关于代数方法的专著。

但是，在第二次世界大战之前，电子技术还不够先进，人们尚未开发出有用的计算机。早期的编程只是后期任务的附属工作，更像是技术人员烦琐的劳动，通常被认为是机器的“设置”或“编码”。真正吸引人的是硬件，因为它被认为是真正的科学和工程。人们普遍认为，电子数字集成器和计算器，即 ENIAC（Electronic Numerical Integrator and Computer），才是开启数字电子计

算时代的关键所在。其实，这台存放于宾夕法尼亚大学的机器只是一台裸机。操作人员必须对这台机器进行手动设置，插拔那些让人眼花缭乱的线缆，并把一排一排的开关放到正确的位置。每解决一个新问题似乎都要重建机器，因为它采用的是硬接线方式。为此，当局雇用了一些具备数学技能的年轻女士，并对她们进行了培训。这些女性程序员称为“计算员”（computer），这个词可追溯至 18 世纪，专门指那些为绘制地图或航海图而编制统计表的计算者。

对 ENIAC 进行编程，使其计算出火炮的弹道轨迹，这是一项艰难的工作，但却是美国国防部指定的任务。为此，这些女程序员想出了一些技巧来简化流程。她们先在纸上画出详尽的图表，并标出在这台机器上可以解决问题的最有效方法。然后，她们再对机器进行手动设置。珍・巴蒂克这样回忆道[5]：“我们清楚地知道如何设置每一条线和每一个开关。”这种操作可能会花费好几个星期的时间。不过，正是由于她们的努力，ENIAC 的公开演示才获得了巨大成功：一条炮弹轨迹用很短的时间就能计算出来，比炮弹本身的飞行速度还快。巴蒂克回忆道：“那简直棒极了，是我一生中最激动的一天。”[6]当时已是战后，1946 年的春天。

对于新兴职业的从业者来说，他们的职业称谓变化很快。人工的“计算员”（computer）变成了“编码员”（coder），然后，这个平淡无奇的称谓又被“程序员”（programmer）势不可当地取代了。事实上，这是英国从业者的贡献。很明显，新称谓听起来更有身份，也更具文学气息。葛丽丝・霍普是软件方面的领军人物，于 1944 年开始在哈佛 Mark I 上从事战备方程式的计算。她一直认为，用“编程”（programming）这个词来形容初期的工作过于高尚。“直到从英国传过来，‘程序员’这个词才开始广泛使用。”她回忆道，“事实上，我认为，书写机器代码的过程就是编码，我们应该留着‘编程’这个词来描述更高一级的工作。但是，它是从英国传过来的，而且比编码员要好听得多，于是每个人都想被称为程序员。”[7]

由于计算机设计上的突破，更高一级的编程很快就成为现实。这个想法最

初源于 ENIAC 工作团队，随后，1945 年 6 月发表的由约翰 · 冯 · 诺依曼撰写的文章“关于 EDVAC 的报告初稿”（A First Draft of a Report on the EDVAC）对其进行了详尽的阐述。作为著名的数学家和博弈理论家，冯 · 诺依曼受聘担任原子弹开发项目“曼哈顿计划”的顾问。原子弹的设计需要成千上万次的计算，当时主要是由大量计算员借助台式计算机来完成的。计算机的潜力引起了冯 · 诺依曼的兴趣，于是他在 1944 年成为了 ENIAC 项目的顾问。后来，在 ENIAC 的基础之上，经过改进，又有了 EDVAC（Electronic Discrete Variable Automatic Computer），即电子离散型变量自动计算机。除了冯 · 诺依曼，还有很多人参与设计了 EDVAC，其中最为著名的便是 ENIAC 项目负责人 J. 普雷斯普尔 · 埃克特和约翰 · 莫克利，但由于最终负责撰写报告的是冯 · 诺依曼，因此他获得了设计“存储程序式计算机”的殊荣。这种设计思路就是后来广为人知的冯 · 诺依曼架构。事实上，现在所有的计算机使用的都是冯 · 诺依曼架构。

第二次世界大战后，早期的存储程序式计算机才真正开始出现。存储程序的理念在于，不仅计算机的数据——当时用于计算的主要是数字——而且程序指令也会存储在机器中。从某种程度上来说，这无疑能够提高效率，并能实现计算自动化。由于程序指令可以设置到打孔卡片或纸带上，与将要处理的数据一起存入计算机中，所以，再也不用手动设置开关和线缆了。

然而，存储程序这个概念还有更深刻的含义。用计算机科学家巴特勒 · 兰普森的话来说，它使软件构建成为了一门“独特的自引用”[8]工程学科，因为所有的计算机制都可以应用在自身中。也就是说，存储程序式计算机可使程序修改其他程序或创建新的程序。正是有了这种以计算机为中介的编程交互方式，如今的计算机编程语言才远远超越二进制 0 和 1 的组合，更易于人类理解。这种编程交互方式相当于计算机内部的数字生态环境：一段代码迅速跳到进程外去修改其他代码，而后者又会循环回来与其他代码混合。无论是电脑游戏、互联网，还是人工智能，所有这一切都源于代码的这种组合、重组以及持续地自我修正的能力。

早期存储程序式计算机的开发人员最早体会到了编程的复杂性，而且这种复杂性常常是无法预料的。剑桥大学的莫里斯·威尔克斯率领的团队制造了第一台存储程序式计算机，并投入运行。这台机器称为 EDSAC（Electronic Delay Storage Automatic Calculator，电子延时存储自动计算器）。在回忆录中，威尔克斯还清楚地回忆起第一次认识到 bug 注定是程序员的克星时的情景。他这样写道："到了 1949 年 6 月，人们已经意识到，要得到正确无误的程序并不像看起来那么容易。"[9]当时，威尔克斯正在研究他的第一个"重大项目"，而当他在剑桥大学的事业更上一层楼时，他回忆道："我突然强烈地意识到，我接下来的时间都要花费在寻找程序的 bug 上了。"

计算机投入应用很长一段时间之后，"软件"这个词才出现，可见人们对这种相当麻烦的技术只是勉强认同。1958 年，在《美国数学月刊》上，"软件"作为计算机术语首次在出版物上使用。[10]普林斯顿大学的数学家约翰·杜奇在文中这样写道："如今的'软件'已包括精心设计的解释路径、编译器以及自动化编程的其他方面，对于现代电子计算器而言，其重要性丝毫不亚于那些由晶体管、转换器和线缆等构成的'硬件'。"但这样的观点在当时并不普遍。

在当时的计算机工程文化氛围下，程序员曾长期被硬件人员轻视。后者认为程序员只不过是从事计算行业的一群放荡不羁的人，硬件才是真正的学科，因为硬件人员大都来自发展较完善的电子工程领域。那时，各个大学都已开设电子工程系，而且硬件的表现也符合物理、化学之类"硬科学"的严谨准则。虽然有些数学家也对计算机和编程着迷，但是他们往往沉迷于理论的高度，而不是编写代码和排查程序。直到 20 世纪 60 年代，随着计算机科学系的成立，编程才被学术界正式认可，并从此稳步发展。

在 20 世纪 50 年代以及此后的一段时间里，程序员的招聘和雇用几乎没有任何科学性可言。那时亟需编程技能：新员工必须经过培训，但是并没有可靠

的方法测试其是否具备编程技能。“在报纸的广告里发现某个人，还没了解清楚，你可能就已经决定要雇用这个人了。在编程的早期就是这样，就像是在讲故事，”在 20 世纪 50 年代后期担任 IBM 编程研究部经理的罗伯特 · 贝莫这样说道，“我们就好像直接从大街上招人回来一样。”[11]刚刚从瓦萨学院毕业的洛伊丝 · 海波特于 1955 年加入 IBM，成为 FORTRAN 编程语言开发团队的一员，该团队共有 10 人。她回忆道：“任何一个人，如桥牌玩家、象棋手，甚至是女性，只要看起来具备解决问题的能力，就会被他们招入麾下。”[12]作为 IBM 的经理，贝莫广泛撒网招揽人才。“我曾经决定以登广告的方式招募象棋手，因为我认为他们会成为相当出色的程序员。这招非常奏效。我们甚至招募到了全美象棋冠军亚瑟 · 比斯盖尔。此前，他大部分时间都在下棋，几乎没编过程序。”但是，事实证明，在编程领域，象棋手并不比常人更出色。1957 年刊登在《纽约时报》《洛杉矶时报》以及《科学美国人》杂志上的广告，最终帮助贝莫招到了四五个合适的人，收获还算不错。贝莫估计，当时美国大概有 1.5 万名专业的程序员，约占全世界代码编写人员的 80%。

如今，这种情况已大为改观。软件行业有了巨大的发展，全球拥有近 900 万名专业程序员。[13]在学术界，计算机科学已经成为受人尊重的领域，科研人员投入大量研究资金来探索软件的奥秘。我们有充分的理由相信，软件不仅赋予了个人电脑和互联网活力，还推动了电话、信用卡网络、机票预订系统、车辆燃料加注系统以及厨房用具等领域的发展。1999 年，总统的科技顾问团队发现，软件已经成为“信息时代全新的物理基础设施”，是一种不可或缺的原材料，“对经济发展、科技研究以及国家安全至关重要”[14]。

的确，现代经济建立在软件基础之上，而且这种依赖性还会继续增强。商业周期和华尔街的狂热周而复始，转瞬即逝，但对软件的需求却从未间断。程序员就是信息时代的技师、工匠、砌砖工和建筑师。这些在早期都是难以想象的，因为没有人能预见迅猛的技术变革会带来什么——依赖硬件和软件的进步，计算领域得到了前所未有的拓展。约翰 · 冯 · 诺依曼和赫尔曼 · 高德斯汀

是当时计算机领域的卓有远见者，他们在 1946 年写道，大约 1000 行的程序指令是“解决目前可以想见的复杂问题的合理上限”[15]。如今，一个电动牙刷就可能有 3000 行代码，而个人计算机程序则有数百万行的代码。

抛开其重要性不提，对于大多数人来说，计算机编程仍然很神秘。这不足为奇，毕竟，软件几乎是完全看不见、摸不着的东西，无法被人感知。但正是软件使计算机能够完成那些有用的、有趣的、令人愉悦的事情。计算机只不过是机器，它很强大，但也很愚蠢。在它们的世界里只有 1 和 0、开或关。20 世纪 70 年代的视频游戏 *Pong*（《乒乓球》），用两条光束作为“球拍”，击打一个像光标一样的“球”，运行这款游戏的简单计算机所能看到的世界是这个样子的：

0011101010101000011100011010101000

而曾在 1997 年击败世界象棋冠军加里·卡斯帕罗夫的 IBM 超级计算机“深蓝”，它所看到的世界也是这个样子的：

0011101010101000011100011010101000

从本质上讲，这两台计算机只有两点区别。一是涡轮增压的位处理引擎使“深蓝”在速度和动力上占有优势，再就是软件。软件是人类智能的体现，是人和机器之间的中介，把我们的问题或命令传达给计算机。

作为一个专业，编程是艺术、科学和工程的奇妙结合。软件开发仍然是一项异常艰苦、循序渐进的工作，它更像是手工艺，而不是机器魔法，它是一种以软件为中介的创造形式。正如厨师烹饪食物，画家创作油画一样，程序员编写代码。不过编程仍是一种实践性很强的艺术形式，热爱它的人们都会为程序有效运行的魅力以及创造事物的渴望所深深吸引。

还是个孩子的时候，葛丽丝·霍普就会拆装闹钟；Unix 操作系统的创始人肯·汤普森，在后院制作了火箭；电子表格软件的合作开发者丹·布里克林，用希斯套件（Heathkit）组装了家用电视机；Java 编程语言的发明者詹姆斯·高斯林，在他祖父位于加拿大卡尔加里市的院子里，改装了旧农机。对那些天生迷恋编程的人来说，创造事物似乎能带来真正的快感，软件尤其能带给他们这种体验，因为它是没有实体约束的中间体。程序员不用钢筋、玻璃和混凝土，就能够建造模拟的城市；不用铝、喷气发动机和轮胎，就能够制造模拟的飞机；不用光、热和水，就能够模拟天气。程序员可以用计算机把想法变成现实，至少是视觉上的真实感，并在自己创造的虚拟世界中对其进行测试。

计算机编程的大部分历史可看成是努力扩大其使用范围的过程，即让更多的人能够更容易地编程。FORTRAN 是第一个真正意义上的编程语言，旨在使科研人员和工程技术人员轻松自如地编程。COBOL 则是为了让商务人士能随心所欲地进行编程。经过多年的发展，编程已经不像原来那么繁重困难了。不过，理想的情况是让每个人都能编程。早在 20 世纪 60 年代，就有人提出这种愿景，但至今仍未实现，虽然编程已取得了长足的进步。现在，几乎每个人都会使用计算机，数万甚至数百万人都能够完成诸如制作网页或者在电子表格中创建财务模型所需要的基本编程。

然而，更重要也更值得一提的是，迄今为止，大部分编程仍然局限于精英阶层。目前，人们已针对熟练的程序员开展了研究。研究表明，他们在智力上的确具备某种特质。他们属于那种对工作之外的事情同样怀有浓厚兴趣的人。例如，对科幻小说感兴趣的程序员会特别关注一两个作者，对音乐、休闲活动或者其他方面感兴趣的程序员也是如此。这种较高的智商和高度的专注正是编程所需要的。心理学领域的研究人员在研究心流时曾观察过软件程序员。“心流”状态指的是，注意力高度集中，全身心投入，心智水平达到最高。这种身心合一的境界就如同运动员进入“状态”。

不过，这类研究只能表明如何才能成为天才程序员以及哪些人具备这样的

潜质。“某些人确实非常适合做程序员，他们比其他具有同等教育水平和同等智商的人在这一方面好很多倍，”莱斯大学的计算机科学教授肯·肯尼迪这样说道，“现在我们还不能真正理解这种现象。”[16]这进一步表明，编程不仅是一门科学，还是一门艺术。

斯坦福大学的荣誉教授高德纳终生都在教授学生掌握这项技能。计算机科学能够成为一门学科，高德纳功不可没。他以编写软件方面的权威著作《计算机程序设计艺术》而闻名于世。他从 1962 年开始动笔撰写，目前已出版了三卷。[17]他的家在斯坦福大学后面的山丘上，二楼的书房里摆着一排排的书。高德纳在书房里说道：“确实有少部分大学生——大概 2%左右——具备异于常人的心智，擅长计算机编程。他们精于此道，智慧如同泉涌……只有这些人才能让计算机做到令人惊叹的事情。我希望这不是真的，但是事实就是如此。”[18]

本书讲述的就是这些具有特殊心智的极少数人的故事，他们能够利用代码创造出神奇的世界。本书只是代表性地——绝非结论性地——对计算机编程历史进行回顾，主要讲述那些著名人物的生平事迹以及他们开发的软件。

第2章

FORTRAN：早期的“转折点”

1952年8月，IBM的新型豪华计算机——国防计算机（Defense Calculator）已经准备好接受试验。包括洛斯阿拉莫斯核武器实验室、道格拉斯飞机公司、洛克希德飞机公司在内的6家客户早已预订了这款机器，而且他们还被邀请到IBM的波基普西工厂，第一时间目睹这台机器的风采。计算机用于计算尚处于初级阶段，是继实验室试验之后迈出的第一步。对这种电子庞然大物感兴趣的主要是美国国防部及相关保密部门，还有新兴的航空企业。他们主要想借助这种庞大的机器使乏味的科学计算过程实现自动化。当时，这项工作是由大量办公室职员用台式计算器手动完成的。后来，人们才逐渐意识到，这种计算机的性能要远远超过那些大的加法机。如果程序编得好，这些计算机完全可以用于

探索新的知识领域。

国防计算机的研制始于朝鲜战争。1950年，朝鲜战争爆发，美国迫切需要生产新型飞机和新型武器。当时，第二次世界大战刚刚结束5年，这些装备的设计和生产意味着对工程计算的需求再度激增。美国国防部及其企业供应商都是财大气粗的客户，但是像他们这样的买家数量并不多。况且，电子计算机不知疲倦的计算能力在国防部门之外是否还有获利的空间也尚未可知。

对此，IBM内部有两种意见。以小托马斯·沃森为首的拥护派、科学家和年轻管理人员认为，对计算的需求将会广泛扩展。雷明顿·兰德公司已经把一台UNIVAC计算机卖给了人口调查局，这就是很好的佐证。持怀疑态度的一派，包括IBM的董事长老托马斯·沃森以及大多数资深管理人员，担心客户太少，而且制造这样极具技术挑战性的机器将会耗尽公司的工程资源。1951年年初，研发新机器的计划获得了批准[1]，不过，鉴于内部怀疑派的意见，这台机器取名为“国防计算机”，表明这是为战争服务的特殊项目。

虽然取名为国防计算机，但它是一台存储程序计算机，因此也是一台通用的机器，一旦收到程序指令就能开始处理各种问题。实际上，这台机器在1953年4月面世时，已经改名为IBM 701，成为IBM 700系列的首台机器。该系列为IBM奠定了世界主要计算机生产商的地位。按照当时的标准，701属于小巧时尚的机型。整个系统由一些独立的单元集合而成，看起来就像百货公司陈列的20世纪50年代的厨房家电：一对纸带阅读机像巨大的橱式电视机，打印机像个烤箱，阴极射线存储器就像冰箱。1952年夏天，当那些特邀观众在波基普西工厂一睹701计算机的风采时，其运行速度给他们留下了深刻的印象。他们带来试样程序，编码之后在纸带上打孔。“他们都被这台计算机惊呆了，”[2]当时在场的IBM的高级管理者卡思伯特·赫德这样回忆道，“刚把程序输进计算机，计算结果就出来了……我们都坐在那儿，说，怎样才能让这台机器一直忙下去？它简直太快了。我们怎样才能做到？”

毫无疑问，20 世纪 50 年代，IBM 研发的这台巨大而又昂贵的机器，数据处理能力并不高，甚至不及今天手提电脑的 1/10。但在 1952 年，701 却是一台运行速度极快的庞然大物。因此，IBM 陷入左右为难的境地——计算机软件和计算机硬件是两个截然不同的领域，但二者又必须相互依赖，正如太极中的阴和阳，缺一不可。要回答赫德关于如何才能让这台高速计算机一直忙下去的疑问，其实很简单，那就是把更多的问题输入机器。但是还有一个瓶颈，那就是编程。

要把工程或科技问题输入计算机，准备工作既艰难烦琐又枯燥乏味，可能要花好几个星期的时间，而且还需要专门的技能。只有很少一部分人具备这种与机器对话的神奇能力，就像原始社会的大祭司一样。然而，这些祭司里面也有离经叛道之人，年轻的程序员约翰·巴克斯就是其中之一。巴克斯曾在“与机器的较量”中受挫，于是他迫切地希望能加快速度、简化编程。“我猜一定有更好的办法，让编程变得更容易。”[3]将近 50 年之后，在位于旧金山可以俯瞰金门大桥的家里，巴克斯如此回忆道。

1953 年年末，巴克斯给赫德写了一封简短的信，请求赫德允许他寻找一种更好的编程方法。得到赫德首肯之后，这个研究项目开始启动，并最终在 1957 年取得了历史性的突破，发明了名为 FORTRAN 的计算机编程语言。项目组的管理一直都很宽松，工作环境也很随意。尽管这个研究项目不断扩展，完成时间一次又一次地推迟，但巴克斯从未做过正式的预算。FORTRAN 的研发团队是逐步建立起来的，成员一个接一个地加入，最终达到 10 个人。[4]这是一个年轻的团队，当 FORTRAN 正式对外发布时，他们才 20 多岁或者 30 岁出头。由于当时大量的计算都涉及数字分析、数学运算和分类整理，所以整个团队都要接受数学方面的强化训练。

不过，他们仍然是别具一格的一群人，有晶体学家、密码员、象棋高手、

从联合飞机公司借来的雇员、麻省理工学院的研究员，还有刚刚从瓦萨学院毕业的女大学生。这些人在一个开放的环境里共同工作，办公桌并排摆放在一起。事实上，他们经常在夜间工作，因为只有这个时候才有宝贵的上机时间，对代码进行测试和调试。除了一起工作，紧密配合之外，他们还共度闲暇时光，例如，午餐时一起下象棋，冬天即兴打雪仗。他们彼此了解，友情也日益深厚，熟悉各自的代码以及工作的机器，甚至是机器的每个部件。而且他们当时都是这个行业的门外汉，成功的机会看起来微乎其微。“我们是当时的黑客。”[5]理查德・戈德堡在 76 岁高龄时如此回忆道。

FORTRAN 团队的成功具有双重意义。首先，他们开发出一种像是结合了英语速记和代数的编程语言。这是一种计算机特有的语言，与科学家和工程师日常工作中使用的代数公式非常相似。因此，FORTRAN 向当时那些需要借助计算机解决问题的人们开启了编程的大门。只要稍加训练，他们就再也不用依赖那些计算祭司把自己的问题翻译成机器语言了。FORTRAN 将人与计算机的交流提高了一个层次，更加接近人类，同时更加远离机器。这就是为什么 FORTRAN 被称为第一个高级语言。

此外，更重要的是，FORTRAN 运行良好。它生成的程序的运行效率与那些最优秀的程序员辛辛苦苦手动编写出来的程序一样。正是由于在编程自动化方面的飞跃，FORTRAN 才被人们采用。机器时间是成本高昂的宝贵资源。如果用 FORTRAN 编写的程序运行速度很慢，比手编程序还要消耗更多的机器时间，那么它就不具备经济实用性了。当时人们认为，没有什么能比得上手工编程的效率。然而，IBM 团队巧妙地设计了 FORTRAN 编译器，结果成功地做到了这一点。简单来说，编译器本身就是一个程序，它可以捕获输入的程序中的人类意图，并按照机器可以理解和执行的方式，对程序进行改写。

新版本的 FORTRAN 语言仍然广泛应用于某些科学计算任务中，例如，天气预测、建立气候变化模型以及高能物理等方面的数字分析。直到现在，那些经验丰富的计算机科学家和程序员依然对 FORTRAN 念念不忘，因为 FORTRAN

是他们学习的第一种编程语言。但是后来由于要适应更新的计算类型，所以新开发出来的语言逐渐取代了 FORTRAN。从某种程度上来说，FORTRAN 是编程的源头。无论编程工具更新换代的速度有多快，FORTRAN 给软件世界带来的巨大进步都不可磨灭。其他程序语言都是在 FORTRAN 建立的基础上发展起来的。杰出的计算机历史学家、弗吉尼亚理工大学教授 J. A. N. 李称，FORTRAN 是编程语言及编译器技术（软件的等效晶体管）发展过程中的“转折点”[6]。1969 年在贝尔实验室设计了 Unix 操作系统的肯·汤普森指出：“如果没有 FORTRAN，95%的早期编程人员将一事无成。FORTRAN 是一个巨大的进步。”[7] 或者，正像现在微软的软件研究领军人物吉姆·格雷说的那样：“一切都从 FORTRAN 开始。”[8]

约翰·巴克斯走上计算机科学之路实属偶然。他出生在美国特拉华州的威尔明顿市。他的父亲塞西尔·巴克斯白手起家，本来要成为一名药剂师，后来却转行当了股票经纪人。作为一家经纪公司的合伙人，塞西尔·巴克斯的事业逐渐做大，家境也开始富裕，并颇具社会声望。孩提时候的巴克斯喜欢用自己钟爱的化学仪器做实验。在他差不多 12 岁的时候，另一个小孩不小心把摩托车开到了海里，因为没法再骑便把它扔了，但巴克斯却让这台摩托车起死回生。他回忆起这段往事时还颇为得意，说：“我一直喜欢鼓捣机械类的东西。”此时的他已经一头灰白的短发，略显消瘦，脸上总是一副谦逊的表情。

在 76 岁高龄时，巴克斯还欣然称自己为“小发明爱好者”。他坦承自己痴迷于掌上控制器，开玩笑说“离了它就没法儿活”。他给大门和车库门加装了自制的远程自动遥控装置。此外，他还自己摸索着做了一个电视机顶盒，仅用一个大容量的计算机磁盘，加上巧妙的编程，就能让看电视的人跳过广告节目，在看现场直播时可以暂停，而且基于数据库搜索还可以记录下电视节目。“这是个很棒的发明，”巴克斯高兴地说道，“它将改变电视行业！”

巴克斯不是个听话的学生，与家人的关系也复杂而紧张。父母把他送到封闭的私立高中，即宾夕法尼亚州波茨敦市的希尔学校。他的学习成绩很糟，因此每个暑假家人都要把他送进学习营，让他预习接下来的秋季课程。“考试不及格就意味着我可以不回家了，这才是我喜欢的。”[9]巴克斯说道。对他来说，在希尔学校待着就是要解决各种问题的挑战，“在那里的乐趣就是打破一切规则”。上了大学，他对待正规教育的态度依然没有任何改变。因此，在弗吉尼亚大学只呆了两个学期，他就因成绩不好退学了。

他在学习上的平庸表现，并非因为智力低下，随后的军旅生涯很快就证明了这一点。1943 年，从弗吉尼亚大学退学后不久，巴克斯就应征入伍了。由于在军队的能力测试中表现出色，巴克斯首先被选送到匹兹堡大学学习工程课程，后来又到哈弗福德学院（Haverford College）学习医学预科的课程。虽然觉得学习医学很枯燥，但是他之后还是参与了纽约医学院在曼哈顿的一个由政府资助的项目。“学医好像就是死记硬背操作步骤和身体部位。”他回忆道。

随后，在决定何去何从的一段时间内，巴克斯，这位古典音乐发烧友，觉得他在曼哈顿的小公寓需要一套好的音响系统。他开始构建自己的高保真音响设备，并且很快参加了学校里一个为无线电技师开办的课程。为了做出扩音器，巴克斯必须计算出声波曲线上的点。他发现解决数学问题虽然艰难却引人入胜。他说：“做这样的计算叫人发怵，但不知为什么也多少引起了我对数学的兴趣。”于是他申请了哥伦比亚大学。鉴于他复杂的学习经历，校方同意他成为试读生。结果，他在哥伦比亚表现优异，不仅完成了学士学位的课程，还在 1950 年获得了数学硕士学位。

就在从哥伦比亚大学毕业前的一个春日，巴克斯参观了位于第 57 大街和麦迪逊大道交汇处的 IBM 总部。之前他听说那台巨大的科学计算机器就陈列在那儿，出于对机械装置的痴迷，他想去看一看。IBM 把这台计算机安放在一

楼，以方便那些好奇的游客从大街上就可以看到它。数千只闪光的灯管，噼啪作响的开关，打孔的卡片发出哔哔的声音，纸带呼呼地转动……这台奇妙的电子装置让无数路人为之震撼，装置虽复杂但设计很精巧。过往的行人可能并不清楚这台计算机是由什么做成的，却难以抑制心中的冲动，纷纷要给它取个名字，年复一年，从未停止。最终他们称之为“Poppa”。

巴克斯很想近距离观察，便鼓起勇气走到里面，一位女士带他简单地参观了一下，并向他介绍了这台名为 SSEC（Selective Sequence Electronic Calculator，选择性顺序电子计算器）的装置的各个零部件。他向这位女士提起他是哥伦比亚大学的数学专业研究生，现在正在找工作。于是，她说可以带他直接上去见 SSEC 的联合发明人罗伯特·雷克斯·席伯。巴克斯记得他当即表示反对，说：“我没打领带，外衣袖子上还有个洞，而且我对计算机一窍不通。”那位女士却坚持说这不是问题，于是带他上楼去见席伯。简短的寒暄之后，席伯问了一连串的问题，巴克斯把这些问题形容为“脑筋急转弯”，诸如怎样用 10 位的计算器进行 20 位数字的排列和加法运算。

巴克斯回忆说，这就像一次不计成绩的非正式口头测验。席伯当场决定录用他。做什么工作呢？“程序员，”他耸耸肩，回忆说，“在当时这可是不错的出路。”

巴克斯加入 IBM 时，公司、行业以及计算机技术都处于快速转型的时期。数十年间，诸如 IBM、雷明顿·兰德公司、宝来公司以及 NCR 等厂商凭借生产商用计算器发展壮大起来。大批量生产、现代铁路和汽车运输的兴起，以及全国电话网络系统的发展带动了规模经济的发展。企业的发展也必须顺应这一趋势，在公司大规模增长时期，计算器有助于管理人员对员工工资、库存和销售等进行跟踪。但是，在第二次世界大战的推动下，技术和计算器生产商超越了商用的电动机器，快速转向了适用于航空和国防市场的电子工程。

SSEC 被 IBM 称为超级计算器，它反映了上述趋势。但这台独一无二的机器，实质上只是 IBM 的一个科研项目，仅设计制造了一台，旨在帮助 IBM 的

研究人员突破电子计算器的极限并获取经验。

SSEC 并不是存储程序计算机，但在 1948 年，它刚刚制造完成时，却是最先进的，也是当时计算能力最强的机器。然而，老托马斯·沃森十分在意“computer”这个词，因为它常用来指从事计算工作的人员（即计算员），沃森担心使用它将会引起公众的恐慌，害怕这种新技术会导致他们失业。由此看来，老沃森不愿意使用“计算机”（computer）这个词是可以理解的。公众的这种担忧一直持续了多年。1957 年，由斯宾塞·屈塞和凯瑟琳·赫本主演的《电脑风云》是一部深受观众喜爱的电影，用浪漫喜剧的手法展现了计算机在一家大公司里引起的不安。伯尼·沃森（赫本饰演）坚持认为，一台名叫 EMERAC 的计算机会取代她部门的员工，而理查德·萨姆纳（屈塞饰演）不仅负责看管这台机器，还负责削减人员。可想而知，结局是伯尼的担心毫无依据，20 世纪 50 年代的电影常以这种方式收场。

巴克斯是从 SSEC 开始编程的。他编写的程序主要用于大规模科学计算，例如计算出数年间任意时刻月亮及其附近行星位置的程序，这需要无休止地处理系数。巴克斯回忆道：“这是纯粹的科学。”

研究可能是崇高的，但编程却像一场阵地战。就像战时的 ENIAC 一样，SSEC 每接受一个任务，都必须重新配置。当研究人员用数学方法解决难题以后，还必须将其输入到这台机器中。这就需要设计复杂的流程图，使计算能够以通用的形式输入机器。接下来还要在预先打印的纸张上逐条按照指令拟定好计算的步骤，这是一项艰难而又烦琐的工作。然后，手动设置每个批次的计算，比如哪些开关要关闭、哪些电线要接入哪些电路，才能使这台超级计算器重新开始工作。

设计 SSEC 上的大型程序可能需要几个月的时间，随后又要在机器上运行 6 个月。平均每 3 分钟它就可能会停止运行，需要编程人员做进一步的处理。

巴克斯说：“作为程序员，你一刻也不能离开。”如果出现问题，可能要断开机器上数以千计的灯管，而且还要查阅二进制编码的数字才能找出解决问题的线索。

调试机器还要借助听力。回路的断开和闭合靠的是继电器，它是装在弹簧上的金属棒，在电磁推力的作用下才能弹起。数千个继电器以不同的排列顺序安装在回路中，时常发出震耳欲聋的声响，但并不是不规则的工业噪声。从机器的某个角落发出的重复节奏，在训练有素的程序员听来像听破录音带一样别扭，这就意味着程序在某个计算环节卡壳了。后来，当新一代 701 国防计算器问世时，无声的电子开关代替了机械继电器，巴克斯回忆说有一种惶恐不安的感觉：“我想知道，怎样才能为这个安静的大怪物排查错误。”

说起自己和这台超级计算器较量的那些日子，巴克斯充满了对旧时光的怀念之情。“噢，这机器太复杂了，真的，而且那时还没有教科书。这些约束真是挑战……我们有太多的机会来展现自己的创造才能了，几乎每时每刻都在创新。”

在 1953 年写给老板的信中，巴克斯提出了编程项目的建议，并着重强调了其在经济方面的重要作用。首先，当时每台装置通常配有 30 个编程人员，这些编程人员的薪酬成本和计算机本身的成本相差无几（701 国防计算机每月的租金为 1.5 万美元[10]，相当于现在的 10 万美元）。其次，对于编程人员来说，1/4 到 1/2 的计算时间都花在了排查错误上。因此，实际上，编程和检错占了整体成本的 3/4。随着硬件的快速更新和价格的下降，编程成本所占的比例似乎还会更高。这是个大问题，并且会越来越突出。卡思伯特 · 赫德看过这封信后，立即批准了巴克斯的请求，马上启动编程研究项目。巴克斯回忆道：“他明白这个项目的重要性。”[11]

1954 年 1 月，巴克斯迎来了首位新人，欧文 · 齐勒。欧文 · 齐勒是布鲁克

林大学的研究生，1952 年加入了 IBM，从事电子计算器“插线板”的编程工作。当时的计算器由一系列的插线板组成，插线板大约长 28 厘米、宽 21 厘米，上面布满了小孔，可以手动把电线连接到这些小孔上。这又是一种硬件编程。完成之后的插线板看起来就像是板子上长出了密密麻麻的线缆丛一样。齐勒很快就显示出自己在编程插线板方面的聪明才智和非凡才能。他在纽约里弗代尔区的公寓里，详细生动地讲述了从事插线板编程的那段岁月：“可以想见，这是相当枯燥的工作，[12]那些从事插线板工作的人都能意识到，简化编程是一种迫切需要。”因此，当征求他的意见时，他当即同意加入巴克斯的项目团队。

不久，第三位成员哈兰·赫里克加入了这个团队。他是爱荷华州立大学数学专业的毕业生，还是一位出色的象棋手，曾经赢得过美国中西部地区的锦标赛。他还获得了耶鲁大学研究生课程的奖学金，但是他在那里并不开心。当看到一篇有关 IBM SSEC 的文章后，他便申请了编程的工作，并被录用。加入 FORTRAN 团队时，他已经拥有 5 年为 IBM SSEC 和 701 机器进行编程的经验，这段经历使他成为了一个毫无激情的编程老手。在 IBM 内部，赫里克被称为天才程序员，对于 FORTRAN 取得的成功，他功不可没。然而，刚开始的时候，他却最持怀疑态度，因为他已经完全沉浸在当时的人工编程实践之中。当巴克斯告诉他这个项目时，赫里克有点不屑一顾。“我说：‘约翰，我们不可能让一种语言像真正的程序员那样生成机器代码并达到像我这样的程序员的效率。’”[13]赫里克在 1982 年这样回忆道，“我可是个很棒的程序员，你不知道吗？”

由于编程工作对智力有很高的要求，当时的程序员根本不相信，甚至还有些鄙视用编程语言代替他们工作的想法。程序员必须精通二进制的机器语言。试一试最简单的数字转换，二进制的机器语言 01 代表 1，10 代表 2，11 代表 3。而 100 表示 4，因为 4 是 2 的平方，需要在第三列加一位。这些列向左移一位，代表 2 的乘方或倍数。8 是 2 的三次方，因此，1000 代表 8（2×2×2）。同样，100000000 代表 256，即 2 的八次方。起初，人们觉得 1 和 0 的二进制体系过于难懂，有些“不自然”。不过，部分原因也在于我们已经完全习惯了基于 10

的数字体系，即每个数位都是 10 的乘方。基于 10 的体系称为十进制，之所以令人感觉舒服，是因为它与人类天然的计数工具是一致的——10 根手指就是我们的数字。

程序员开始使用二进制的早期简化工具是八进制。八进制是基于 8 的数字体系，包含 8 个数字（0、1、2、3、4、5、6、7），其数位左移表示 8 的乘方。采用八进制是因为对人来说它相对容易读数，与二进制相比也更容易，而且还能转换成机器的二进制。原因在于，再次强调一下，8 是 2 的倍数。对于早期的程序员来说，八进制是第二自然的方式。“我们曾开玩笑说要用八进制开支票。”[14]FORTRAN 团队的成员洛伊丝·海波特说道。甚至还有关于八进制的段子：“为什么程序员分不清圣诞节（12 月 25 日）和万圣节（10 月 31 日）？因为十进制的 25 等于八进制的 31。”[15]这个笑话的奥秘在于，如果写出来的话，就变成了 Dec（imal）25=Oct（al）31①。（八进制的 31，也就是 3×8 加上 1，等于十进制的 25。）20 世纪 60 年代以来，由于计算机和软件越来越大、越来越复杂，当程序员确实需要从机器的层面理解问题时，他们普遍采用基于 16 的计数系统，即以十六进制作为二进制的速写法。十六进制使用的符号是 0~9 和 A~F。

减少编程难度的进一步措施是开发“汇编”程序。这样程序员可以用助忆缩写的方式来书写指令，例如可以用 LD 表示 load，或 MPY 表示 multiply，后面再加一个数字指代计算机内存中的某个位置。然后，用小的汇编程序翻译这些象征性的符号指令，或者说“汇编”成二进制，这样机器就可以执行这些指令。这些速记符号、缩写和数字的混合语，称为汇编语言。当时每种计算机都有自己的汇编语言，而每台机器的环境就像中世纪的封地一样，有其独特的方言。尽管如此，在通向 FORTRAN 及其编译器这样的高级语言的道路上，汇编语言以及汇编程序仍是至关重要的步骤。

---

① Decimal 意为十进制，而 12 月是 December，均简写为 Dec；Octal 意为八进制，而 10 月是 October，均简写为 Oct。——译者注

汇编语言最早出现在英国，剑桥大学的 EDSAC 是第一台运行的存储程序计算机。剑桥大学的编程创新也是出于同样的考虑：鼓舞 FORTRAN 团队以及后来的软件开发人员。“最初的目的是让人无需经过专门的培训就能轻松地使用。”[16]戴维·维勒回忆道。他在 1948 年秋加入剑桥项目组时还只是个 21 岁的研究员。维勒负责为 EDSAC 编写汇编程序，他称之为“原始命令”，即 30 行巧妙而又简洁的指令。原始命令被翻译成了用简单的汇编语言写成的二进制指令。例如，告诉计算机“把存储器位置 123 中的数字加上 F”的单行指令，写出来就是如下的样子：

A 123 F

在 1951 年出版的第一本编程教科书《数字电子计算机的编程准备》中，剑桥项目组讲述了他们的工作。该书的作者莫里斯·威尔克斯、戴维·维勒和斯坦利·吉尔决定首先在美国出版该书，因为美国拥有更多的编程人员，所以他们的工作成果可以产生最大的影响力。此外，该书还描述了“子程序”的用途，即程序中经常用到的部分，它们可以被保存在“库”里，供很多应用软件在需要的时候重复使用。剑桥项目组提出的代码复用概念，直到今天仍然是减少软件 bug 及提高程序员工作效率的主要方法之一。

在巴克斯的坚持下，FORTRAN 团队设定的目标远远超出了汇编语言。每一行汇编语言代码会转换成一条二进制机器代码的指令，对于 20 世纪 50 年代的汇编语言程序员来说，编程是每次一行的工作。巴克斯希望打破这种一对一的编程模式，实现一行程序代码能够转换成很多条机器指令。他的计划如果成功，将不仅带来计算机技术上的进步，而且还会促进文化的转变。归根结底，他的目标是使汇编语言程序员的高超手工技能实现自动化。

其他人也在为同样的目标而努力，希望减少计算对于编程祭司的依赖。也许最坦率的变革拥护者要属葛丽丝·霍普。20 世纪 50 年代她在雷明顿·兰德公司的 UNIVAC 项目组工作。她对“编程问题”的看法与巴克斯非常相似。“我觉

得总有一天要转变观念，不应该逼着人们去学习如何为计算机编写代码，”[17] 1976 年她解释道，“恰恰相反，计算机应该学会如何回应人，因为我们不可能教所有美国人如何编写计算机代码。必须建立一个接口，它能够接受以人为导向的事物，然后利用计算机将其转换成机器代码。”

霍普经常在计算机会议上呼吁大家支持她倡导的“自动编程”。在这种思想的指导下，她组编了几个软件工具。她为 UNIVAC 编写了一个自动编程系统，把一些代码段组合到单个的程序中，她称之为 A-0 编译器，随后还推出了 A-1 和 A-2 版本。[18]霍普后来成为 COBOL 研制监督委员会的负责人，与其说她是编程高手，还不如说她是技术专家、梦想家和行业女活动家。不过，她的编译器生成的程序运行得太慢，不能满足大多数商业应用的需求[19]。而且，它更像是一个编程的工具集，并不符合编译器的现代定义：把用人类熟悉的语言书写的指令转换成二进制的程序。

首个真正现代意义上的编译器可能是为麻省理工学院一个名为“旋风计划”的政府资助项目开发的。[20]1954 年年初，麻省理工学院的两位研究员 J. 哈尔卡姆比 · 拉宁和尼尔 · 席尔勒编写了一个将数学等式转换成机器代码的程序。同年 6 月，巴克斯和齐勒造访了麻省理工学院，直接考察了这个编译器，并与两位开发者进行了交谈。“非常好，概念设计非常棒，”齐勒回忆说，“不过他们是学院派的做法，不太注重效率。”事实上，要完成同样的计算，麻省理工学院的编译器程序比人工编码的程序要多花差不多 10 倍的时间。

最初的 FORTRAN 三人组——巴克斯、齐勒和赫里克，野心勃勃地制订了目标，要与编码员一争高下，纵然屡遭挫折，也绝不动摇。他们明白，成败取决于编译器的转换效率，这远远比语言本身重要得多。“我们只是不断地编写语言。”巴克斯解释道，“我们认为语言设计并不是什么难题，只不过是个前奏，真正的问题是设计出能够生成有效程序的编译器。”[21]

他们用什么方法设计出了 FORTRAN 可能无关紧要，但 FORTRAN 语言的设计确实为我们留下了宝贵的财富，特别是那个看似平淡无奇的决定：将 Go To 声明作为其基本命令之一。艾兹格·迪杰斯特拉是更加规范的“结构化编程”概念的学院派拥护者。1968 年，他给业界领先的专业组织——美国计算机协会的杂志《ACM 通讯》的编辑写了一封慷慨激昂的信。[22]在“Go To 声明有害无益”这一戏谑式的煽动性标题之下，迪杰斯特拉指出：“程序员的素质与其在程序中使用 Go To 声明的密集程度成反比。”迪杰斯特拉写道，Go To 声明具有“灾难性的后果”，以至于他“确信所有‘更高级’的语言都应将 Go To 声明排除在外”。20 世纪 60 年代末，也就是迪杰斯特拉写这封信的时候，软件程序在规模和复杂性方面已经有了极大的发展。Go To 是一种“无条件跳转”的命令，允许从软件程序的某个位置跳转到其他任何地方，从而改变程序的执行和控制。在迪杰斯特拉看来，当时那种大型的复杂程序中，Go To 命令允许程序员编写运行时可以跳转到其他地方的程序，这会引起“代码混乱”，因此，是导致灾难的罪魁祸首。不过，这并不是 FORTRAN 团队在 20 世纪 50 年代中期面临的大问题。20 世纪 60 年代令人头痛的问题，部分是由于巴克斯团队成功解决了 50 年代的难题所造成的。

FORTRAN 这个名字就是整个项目组热衷于转换器或者编译器的最好证明。它是公式转换系统（FORmula TRANslating）的缩写，公式转换系统后来改名为公式转换器，似乎想要表明它是某种真实存在的而非无形的“系统”。巴克斯在 1954 年提出这个名字时，并没有得到同伴的热烈响应。“F-O-O-O-R-T-R-R-A-A-N”，赫里克一字一顿地慢慢念着，就像不喜欢 IBM 在 1982 年赞助拍摄的纪录片一样，“它听起来像是倒着拼写的名字”。不过，这个名字精确地描述了整个项目，而且没人能想出更好的名字来，于是 FORTRAN 就这样定下来了。当然，FORTRAN 语言也是如此，直到今天还应用于从超级计算机到个人电脑等诸多机器上，这远远超出了巴克斯和 FORTRAN 团队其他成员的想象。FORTRAN 的最初目标只是使程序员在国防计算机 701 的后续机型 IBM 704 上

更加轻松地编程。

1954年11月10日，FORTRAN项目组撰写了一份描述FORTRAN语言及其目标的报告，其标题是“初步报告：IBM数学公式转换系统规范”。欧文·齐勒手里拿着一份原始文件的副本，这是他刚从地下室的纸箱里找出来的。这份报告共有29页，全都是关于FORTRAN语言的数学解释及其使用规则，包括DO、IF、GOTO和STOP等命令。抛开其干巴巴的标题不说，这份报告还算是一份富有敏锐洞察力、间或激情澎湃的市场推介书。按照现代商务的标准来看，它就是一份“愿景陈述”，详细描述了巴克斯对FORTRAN项目的计划和希望，以及对其影响力的乐观预期。即使是现在读起来，这份报告也依然引人入胜，充满质朴气息和远见卓识。报告重申了关于手工编程经济性的论断，即巴克斯在1953年向赫德指出的手工编程成本日益高昂的问题，也是FORTRAN项目得以启动的原因。当时，真正让人们对这个话题产生兴趣的是一段引人注目的文字，开头是：“鉴于FORTRAN几乎会淘汰编码和检错……”[23]编码指的是FORTRAN编译器自动生成的机器代码，因此，那段文字应该是描写性的文字，而非预测性的文字。但是，说FORTRAN能够消除调试，即使在当今的读者看来也无异于天方夜谭。时至今日，对程序员来说，调试依然是痛苦之源。

现在想来，在1954年这份初步报告中，最强有力的经济性论断依旧是赋予编程更多的权力这一主题。报告预言，FORTRAN不仅能提高编程效率，而且还将让更多的人具备编程能力，这正是软件界一直梦寐以求的目标。

报告指出：“如果想有效地操作704，那么，通过使用FORTRAN，一个人所需掌握的知识总量远远少于直接编码……事实上，程序员所具备的数学知识已经包含了大量有关FORTRAN的信息。这样更多的人群都有机会利用704，使其充分发挥作用，而不是像以前那样，只有通过昂贵而又费时的编程训练才能做到。”

带着这份颇具市场前景的文件，巴克斯、齐勒和赫里克开始周游全国，游说那些订购了 704 的客户。他们去了华盛顿、洛杉矶、匹兹堡、阿尔伯克基，以及其他一些相信 FORTRAN 的前景和编程未来的地方。他们原本希望激发人们对这个项目的兴趣，并从用户那里听取一些意见和建议，结果，却无人理会他们的热情。FORTRAN 报告对客户来说只是一厢情愿的想法，在葛丽丝·霍普等人兜售了所谓“自动编程”这一虚假希望之后更是如此。甚至连那些有见识的用户也不太相信，FORTRAN 及其编译器能够以接近手工编程的效率生成机器代码。回忆起那次令人沮丧的行程，巴克斯说：“我们几乎没有收到任何建议或反馈。”[24]因此，FORTRAN 与“客户驱动”需求以及用户参与产品开发的现代商业理念几乎毫无瓜葛。对客户来说 FORTRAN 远在天边，还用不着认真考虑。

虽然失望而归，但这趟客户之旅对 FORTRAN 团队也起到了意想不到的激励作用。“我们认为这是个很好的项目，但每个人都告诉我们不可能实现，”巴克斯回忆道，“我们强烈地渴望证明给他们看。”胜利的曙光终将到来，只是不会那么快。巴克斯回忆，在初步报告之后，他的上司会定期询问 FORTRAN 何时能完成。他总是给出同样的答复：“6 个月。我们确实总是感觉 6 个月之内就会完成，但实际上，前后共花了将近 3 年的时间。”[25]

1955 年，巴克斯开始为团队添加人手，通过各种渠道搜罗新人。巴克斯拜访了麻省理工学院，这是主要的计算机研究中心，和 IBM 的关系很密切。他向该院的数字计算机实验室介绍了 FORTRAN 项目，并询问他们能否派人参与开发 FORTRAN。于是，麻省理工学院派遣了他们的一个明星程序员：谢尔登·贝斯特。巴克斯还充分利用行业内的丁点儿兴趣，从联邦飞机公司借来了罗伊·纳特。纳特的确为 FORTRAN 这个项目带来了惊喜，他是一位非同寻常的程序员，能在脑海中像机器那样“运行”一个程序，随后以超人的速度准确

无误地写下代码。令巴克斯惊奇的是，当纳特觉得已想好如何解决某个软件难题后，他会径直走向打孔机，在没有任何提示的情况下，直接将程序无误地写到打孔卡片上。此外，巴克斯还在 IBM 内部索要他看上的或别人推荐的人才。他想要那种看上去既聪明又具备编程技能的人。“并没有什么正式的形式，”他回忆道，“我们的团队成员逐个增加，自然而然就这样了。”

招募来的人经常缺乏经验。曾在奥地利为美国国务院工作过的密码员罗伯特·尼尔森，就是一名新加入 IBM 的员工，主要负责科技文档的录入等常规工作。但是，巴克斯很快就发现了尼尔森的技术天分。“他很快就成为了一名出色的程序员，绝对是 FORTRAN 项目的关键人物。”齐勒评论道。

理查德·戈德堡拥有纽约大学的数学博士学位，他原本打算从事教育工作。在达特茅斯学院工作了一个学期之后，戈德堡意识到自己并不想做一名教师，于是又搬回纽约，在 IBM 找到了一份工作。“我对计算机一无所知。”他回忆道。但是，戈德堡在 3 个月的编程课程中表现出色，于是被推荐给巴克斯，加入了 FORTRAN 项目。洛伊丝·海波特则是从瓦萨学院毕业后直接进入了 IBM。上学时她“擅长数学和其他理科，而像英语这样的模糊学科的成绩非常差”。作为一名能拿到奖学金的优秀学生，她在数学和其他理科方面异常优秀，暑假都是在贝尔实验室度过的。当她毕业时，IBM 以每年 5100 美元的起薪争取她，这几乎是贝尔实验室工资的两倍。在那个年代，这个薪资对她来说非常高。海波特说：“他们告诉我这是一份计算机编程的工作，我只知道个大概。但是我想，既然他们愿意支付那么高的薪酬，这份工作一定很有趣，很有挑战性。”在编程课上，她证明了自己非常适合这份工作，于是被指派加入 FORTRAN 团队。

对戴维·赛尔来说，计算机起初只是一个工具，后来却成为了他的职业。20 世纪 50 年代初期，赛尔是宾夕法尼亚大学从事生物物理学研究的晶体学家，专注于研究碳氢化合致癌物的分子结构。即便在当时，像这样针对晶体结构的研究也需要计算机的协助。由于学校的计算机运行速度太慢，赛尔开始寻找更

快的机器。他把目光瞄向了 IBM 701，结果却被告知，使用这台机器每小时需支付 400 美元，不过，为了支持他的科学研究，他可以先免费使用几个小时。IBM 的管理层也有市场方面的考虑，所以将赛尔这类工作视为深入科学计算领域的一种途径。于是，赛尔前往纽约，用八进制编码逐行写下了他的程序，并将其输入 701。他发现他写的程序不但能满足他对科研的需求，而且还激发了他对编程的兴趣。“就像进入了一个按预期方式运行的世界，一个以逻辑为主导的世界。”赛尔回忆道，“当你运行程序时，预料之中的事情就会发生；如果它不能按照预期运行，那就有某种逻辑原因。”[26]后来，赛尔加入了 IBM，主要从事科学应用方面的研究，但同时也更深入纯粹地研究了编程。他曾编写过一个诊断程序，用来查找 IBM 704 上的 bug。于是，巴克斯就把赛尔“借”到了 FORTRAN 项目组，而且巴克斯也总有办法留住那些他借来的人。

在计算机科学中，一个很棘手的问题是如何让机器充分发挥作用帮助人类解决难题。巴克斯选择了拆分编译器问题，这种方法虽然难以理解，却是 FORTRAN 名副其实的成就之一，而且像其他创新一样，事后才显而易见。看一下当今大多数语言编译器就会发现，这些编译器的步骤或阶段和 20 世纪 50 年代的 FORTRAN 编译器并无不同，采用的都是巴克斯所使用的问题解决架构。

由此，FORTRAN 编译器及 FORTRAN 项目的其他工作都能被划分成一些可操作的任务。编译器首先对高级语言、数学符号和英语缩写等进行初步扫描，或者叫语法分析。接着，对程序进行复杂的分析，以便编译器能把工作重点放在使程序的核心部分自动化上，也就是程序中重复出现的操作。然后，编译器必须计算出如何用最少的时间分配其编译指令使机器运行。最后，经过编译的程序必须被“汇编”成机器代码。

FORTRAN 编译器完成上述工作所使用的并非蛮力，而是一种优雅、高效的方式，它似乎赋予了程序某种生命力甚至某种智能。1957 年年初，FORTRAN

项目已经进入内部的最后冲刺阶段，当团队拿到经过机器编译的程序时，他们常常感到非常惊奇。编译器将程序进行了编译、对程序表达式和计算顺序做了优化，完成了巴克斯所谓的“惊人的转换”[27]。在仔细核查了编译器所做的改变之后，他们才明白编译器的工作是行之有效的。但是，巴克斯说，编译器采取的一些步骤是“连程序员自己都没有想到的”。事实上，FORTRAN 团队主要观察了一个设计精良的复杂软件的工作，该软件遵循通用的规则并嵌入程序员算法中的特定指令，以完成指定的任务。尽管如此，在看到 FORTRAN 编译器的工作之后，团队成员意识到软件是一个特殊的中间体。齐勒回忆道：“这让我们很惊奇，编程指令之间相互作用，就好像编译器有生命似的。”

人们常说，最优秀的软件设计师和最优秀的程序员往往具备一种超能力，这就是概念化和程序化的思维、高级和低级的工作。FORTRAN 二者兼具，而且能力超强。FORTRAN 语言本身就是高级的、概念化的成就。它不仅使程序员编程更加容易，而且还确保了 FORTRAN 的生命力。虽然最初研发 FORTRAN 的目的只是使其能够在 IBM 704 上运行，但是 FORTRAN 的设计十分巧妙，能够在更高级的机器上运行。因此，FORTRAN 可以不受特定机器环境的束缚。戴维・赛尔指出：“基本机型都可以成为其核心内容，这种编程语言可以使用 50 年。”

此外，FORTRAN 还有力地打破了汇编语言逐行计算的模式。用 FORTRAN 语言写的一行命令可转换成多行机器指令，再次简化了编程的过程。下面的例子[28]是一个初级的 FORTRAN 程序，可将华氏温度转变成摄氏温度。

```
WRITE(*,*)“Please enter Fahrenheit temperature:”
READ(*,*)FAREN
CELSIUS=(FAREN-32)/1.8
WRITE(*,*)“The Celsius equivalent is:”,CELSIUS
STOP
END
```

在汇编语言中，同样简单的程序需要 60 多行代码。FORTRAN 的一个公

式转换行(CELSIUS=(FAREN−32)/1.8) 就可以转变为以下五行汇编语言指令，用汇编语言编写为：

```
fld 32real [0001BEC8]
fchs
fadd 32real [0001E000]
fdiv 32real [0001BEC0]
fstp 32real [ebp-08]
```

然后，如果用机器能够理解的二进制代码，上述一行 FORTRAN 命令就会变成如下的五行：

```
110110010000010111001000101111100000000100000000
110010011110000
110110000000010100000000111000000000000100000000
110110000011010111000000101111000000000100000000
11011001010111011111000
```

为了使 FORTRAN 编译过的程序能够达到和程序员一样的效率，FORTRAN 团队需要长期从事艰苦的基础工作。许多业内人士都认为这个目标根本无法实现。这的确是一项异常艰苦且令人沮丧的工作。多年后，当被问及研发 FORTRAN 时有哪些教训时，巴克斯毫不隐讳地表达了一种观点：创新就是反复尝试的过程。“你要心甘情愿地接受失败，”[29]他解释道，“你要想出很多办法，然后非常努力地工作，结果却发现这些办法都不管用。接下来你还要反复这样做，直到找到一个管用的方法为止。”尽管经受挫折和怀疑，他们依然愿意坚持下去，这很大程度上要归功于 FORTRAN 团队的融洽氛围。这是一个年轻、聪明而且紧密团结的集体，充满活力和乐观精神。在某种程度上他们把自己视为 IBM 的局外人，正在一个全新的领域从事一种前所未有的工作，而且几乎没有规则可循。“有段时间，IBM 对这项新业务的要求非常宽松，”赛尔回忆道，“我们有些像硅谷的运作方式。”

FORTRAN 团队最初称为“编程研究小组”，在位于纽约麦迪逊大道 590 号的 IBM 总部大楼附楼第 19 层的办公室里工作。他们的办公室紧邻这栋大厦的升降机的动力机房，谈话有时会被隔壁巨大的机器声打断。办公室内部设施一直都很简朴，在研发 FORTRAN 项目的 3 年中，它好像都是临时的办公地点，即使在队伍壮大后搬到第 56 大街一幢办公楼的第 15 层之后，依然如此。他们总是离群索居。罗伯特·贝莫的办公室在 FORTRAN 小组的大办公室的对面，他时常回想起 FORTRAN 团队当时的工作状态：“他们在里面埋头工作，不分昼夜。”[30]欧文·齐勒记得有段时间他们要夜以继日地工作。当时谢尔登·贝斯特正被一个特别棘手的问题困扰。为了详细探讨这个难题，齐勒逐渐养成了一个习惯，常在下班后和贝斯特一起去地铁附近散步。他们边走边聊，常常绕着整个街区走好几圈，然后齐勒才走进地铁站，而贝斯特则走回附近的公寓。

不过，他们并非一直都在工作。冬季时，他们很可能在办公室里打雪仗，弹药就是从窗台上挖的雪球。每天他们都会休息一会儿，喝点咖啡，吃些点心，围坐在角落里边吃边聊。他们在吃午餐时玩 Kriegspiel 游戏（军棋，德语）。Kriegspiel 是一种“盲棋”，玩的时候两名选手并排坐着，每人都有一块棋盘，中间有隔离物挡着，看不到对方的棋盘。每位选手依次走棋，并尽力想象对手的棋路。还有一名裁判，负责提供“线索”。当一枚棋子被成功吃掉时，或者一方由于对方的棋子阻挡了道路而无法移动棋子时，裁判就会提醒他们。这种游戏既能训练智力，又能让他们得到休息。

巴克斯以蜂窝状的架构来组织工作。每个小组都由一到三个人组成，每个小组都是一个有自主权的单元，可以自由选用最有利于他们完成工作的任何技术。但是，每个小组必须与其他小组就编程规范达成一致，只有这样，软件代码才能顺利地衔接配合。各个小组都有不同的任务，同时又需要严密地配合协作。洛伊丝·海波特负责“流”分析，从本质上来讲，就是使用被称为蒙特卡洛模拟的数学技术来预测编译器的高流量区域。“我们已经形成了一种氛围，如果你不明白自己的程序错在哪里，可以向旁边的人求助，”她回忆道，“没有

人会担心被别人看成笨蛋，或是将代码据为己有。我们在共同学习。”

齐勒和尼尔森负责最难的任务之一：分析并优化所谓的编译器内循环。内循环的调优是必要的重复性操作，也是提高编译器工作效率的关键。齐勒和尼尔森必须使其在软件内实现自动化，这项工作被程序员视为技术的极限，即强制指令跳出循环。

他们采用了过程式编程方法，对内循环进行仔细的研究，目的是找到效率最高（即机器时间耗用最少）的执行方式。随后，当编译器出现类似问题时（通常情况），他们还要设计出调用这一有效步骤的编程语句（特殊情况）。“我们开发的是一个重复的过程，反复不断地向前推进。”齐勒解释道，“我们不断地尝试保存一条‘加载’指令或‘存储’指令，然后修改执行的命令，这样我们一步步地去除了越来越多的计算量。”[31]

由于机器时间是稀缺资源，FORTRAN 的检错通常都在夜间进行。巴克斯团队将开创性的技术应用在 FORTRAN 编译器上，但他们同时也使用了纸带和打孔卡片这类传统的编程手段。直到后来，这些传统手段才逐渐消失。从 20 世纪 60 年代开始，由于存储器和存储技术等硬件领域的发展，程序员可以用连接到计算机上的键盘来编写代码，再后来就可以直接在个人电脑上编程了。FORTRAN 程序员先是把代码写到条状或网格状的纸上，然后再用打孔机把程序打到卡片上，打孔工作由 FORTRAN 团队亲自完成。接着，再把卡片放到读卡机上，对每张卡片上的数字孔洞进行解码。然后，读卡机把数据和程序指令输送给计算机，程序就可以运行了。（据戴维・赛尔说，由于精确地复制大量的卡片非常困难，所以，在 1957 年年末，FORTRAN 实际上是以磁带的形式送给客户的。）有近一年的时间，FORTRAN 团队在早已变成纽约一景的兰登酒店租了个房间，白天在那睡一小会儿，然后回到附近总部的办公室里，整晚都待在那，争分夺秒地在 704 计算机上试验 FORTRAN 语言，直到拂晓才离开。离开前，他们会把最难解决的错误放到一个文件夹中，表明需要特别关注并做进一步处理。这个文件被戏称为“坟场”[32]，一个充满恐怖和死亡的地方。

IBM 以相对宽松、共同协商的方式管理 FORTRAN 项目。不过，项目组也必须遵守 IBM 的某些规则。在这方面的管理上，巴克斯看上去就像个破坏分子。当时，IBM 推行年度员工考评，称为“绩效提升计划”（Performance Improvement Program），简称 PIP。和现在大多数考评项目一样，PIP 按照死板的公式给出分数和评级。巴克斯认为，PIP 体系并不适合评价他的程序员们的表现，因此他基本上置之不理。例如，某天下午，他叫洛伊斯·海波特过来谈话，谈了她的工作的事，称赞她干得很出色，然后，他从桌子的另一边推过来一张小纸片，说道：“这是你的新工资。”海波特记得加薪的幅度令她非常满意。当她起身离去的时候，巴克斯轻描淡写地说：“要是有人问起来，你就说这是你的 PIP。”

FORTRAN 首次面世于 1957 年 2 月在洛杉矶召开的西部计算机联席会议上。参加会议的主要是那些为数不多的 SHARE 会员，他们所在的公司都使用 IBM 的计算机。SHARE 会员主要有来自飞机制造商、大生产商以及政府实验室的管理人员和科学家，会员之间可以随意分享信息、抱怨，甚至程序，因此该组织被命名为 SHARE。那时，软件还不是一个独立的行业，因此，人们认为编程技术是应该分享的有用知识，而不是被保护的知识产权。

在洛杉矶会议上，IBM 预先安排了一场关于 FORTRAN 的公开演示，也就是今天业界所熟知的“Demo”。在会议开始前不久，IBM 请几位客户提出现实中遇到的烦琐计算问题，诸如喷气机翼设计中的气流计算等。针对提出的问题，汇编语言程序员编写代码，同时也用 FORTRAN 写出代码。当汇编程序员完成之后，这两种程序分别输入 704 中——一个由手工编写，另一个由 FORTRAN 编译。结果证明，FORTRAN 程序的运行效率接近汇编程序，有时与汇编程序一样。也就是说，在解决一个标准问题时，FORTRAN 程序耗用的机器时间并不比手动编码程序多。FORTRAN 由此向世人证明那些怀疑者是错误的。“对人而言，

这是一场革命。”齐勒说，“从这一点来说，我们都知道自己创造了奇迹。”[33]

丹尼尔·麦克拉肯第一次接触 FORTRAN 是在 1958 年，当时他是纽约大学的研究生，并已获得该校的奖学金。那时，他已有 7 年的编程经验，主要为通用电气工作。他记得，虽然不是很确定，他那时用 FORTRAN 编写的第一个程序就是计算液态的热流。他至今还能回忆起当时那种兴奋的感觉。使用 FORTRAN，他能近似真实地反映出要解决的数学问题，并迅速地编写出程序，而不必过于关注为机器编码。

他指出，像他这样的熟练程序员用 FORTRAN 编程的速度比用汇编语言至少快 5 倍，他说：“也许有点儿后见之明[34]，但我在当时就已经意识到这将为更多的人开启计算之门。”他现在是纽约城市大学计算机科学教授。他的直觉后来证明是正确的，而他本人也从日益增多的编程人群中受益颇深。20 年来，麦克拉肯以斯蒂芬·金这个笔名出版了介绍如何编程的系列丛书。《FORTRAN 编程指南》出版于 1961 年，是他第一本获得大奖的图书，销量达 30 万册。

FORTRAN 最大的成就也许在于证明了高级语言是可能的和实用的。它的出现扫除了一个巨大的障碍，开启了软件领域多年来持续不断的创新，同时使编程人员能够更容易地进行计算机编程。现在大多数年轻的程序员，如果一点也不懂 FORTRAN 的话，只会将其视为古董。但是，那些在 20 世纪 50 年代从事过计算的人们，仍然对 FORTRAN 的影响力有很高的评价。约翰·麦卡锡是颇具影响力的 Lisp 编程语言的创建者，他帮助开创了 20 世纪 60 年代的分时计算和人工智能领域。这位斯坦福大学的荣誉教授认为计算机科学研究的现状过于平缓。他认为，目前计算机科学研究过于关注细小的步骤以及缓慢的改进，缺乏富有想象力的思维和突破性的目标。不过，当被问及在早期有什么真正给他留下了深刻印象时，麦卡锡毫不迟疑地答道：“FORTRAN，它给我留下很深的印象。”[35]

第 3 章

20 世纪 60 年代的惨痛教训：从繁盛到 COBOL 和 IBM 360 计划成为现实

1957 年，当 IBM 的客户看到 FORTRAN 时，着实印象深刻，这引起了 IBM 高层的关注。突然之间，他们开始从全新的角度看待这个长期以来几乎被放任自流的项目。不管 FORTRAN 是什么，总之它看起来很管用，有助于解除编程的痛苦，客户为这样的前景感到兴奋。IBM 的管理层明白，这是 IBM 极具竞争力的优势。FORTRAN 能够帮助 IBM 卖出更多的计算机。但是在 IBM 内部，并不是每位员工都如此热衷于 FORTRAN。弗朗西丝 · 艾伦很快就认识到了这一点，她于 1957 年 7 月加入 IBM，当时年仅 24 岁。她接受的第一个任务就是教公司的程序员学习 FORTRAN，这可是个艰难的任务。“在 IBM 内部存在巨

大的阻力，”艾伦回忆道，“程序员都坚信，没有一门高级语言能像汇编语言那样完美地完成工作。”[1]

多年来，应用最广泛的编程语言都具备两个特征：一是能够满足业内在特定时期的特殊需求，二是致力于使程序员更加轻松地学习。随着程序员队伍不断壮大，当他们再去学习一门新的编程语言时，最耗费时间和金钱的就是培训成本，经济学家称之为学习某种新事物的“转换成本”。很明显，FORTRAN解决了这个行业难题，它将一些英语单词与代数符号相结合，从而使这种编程语言让程序员倍感亲切。但是，当时的程序员对二进制和汇编语言的掌握来之不易，学习 FORTRAN 的“转换成本”似乎有些高。要他们接受一项新技术，就意味着他们自身技能的贬值。为了说服自己的程序员，IBM 发布了一条公司命令，宣布到 1957 年 9 月，IBM 新研发的机型——704——将采用 FORTRAN 来编程。“IBM 之所以强迫程序员使用 FORTRAN，确实有业务和市场方面的动因，”艾伦回忆道，“如果连我们自己都不用，又怎么能要求客户使用 FORTRAN 呢？”

强制措施非常有效，借助 FORTRAN，人们解决了计算领域一个又一个难题，编程实践开始快速地发展，并开创了一个长达 10 年的繁荣时期，这让人感觉前景无限广阔。随后，Lisp、Algol、COBOL、Snobol 以及 PL/1 等新的编程语言层出不穷。一个新的研究分支“人工智能”——使计算机编程与人类的智慧相匹配——开始建立，并获得了一定的声誉。

人们逐渐意识到软件不同于硬件，编程也开始成为一种独立的职业。但它还处于萌芽阶段，没有标准，没有资质，也没有学校能够传授系统的理论知识。无限乐观甚至无知的早期阶段于 20 世纪 60 年代后期结束。这时，人们才痛苦地发现，开发大的软件系统要比预想的更加困难，花费的成本也更大。编程这种职业逐渐成熟，软件开始被看成是一种生意。早期的那些经历塑造了当时计算的目标，即测试出已有技术的种种限制，并据此确定要解决的基本问题。

编程早期的大胆举措成就了 FORTRAN 的前指导讲师弗兰·艾伦。她在

IBM 接受的第一个任务就是为美国国家安全局的超级计算机 Harvest 编写软件，Harvest 是冷战期间一台专门用于通讯监视和破译电码的计算机。此后，她便很快得到了晋升。最后，艾伦成为 IBM 的一名研究人员。2001 年，她为公司的超级计算机 Blue Gene 编写软件。Blue Gene 是生物计算领域的新型计算机，用于探索最基础的生命过程之一——蛋白质折叠的奥秘。据她回忆，20 世纪 60 年代的计算经历使她“对于一个人能够做什么非常乐观”。当时，很多人都赞同她这种观点，甚至有人预言，计算机即使不能超越人类的智慧，也将很快赶上人类的智慧，胜过那些被认为拥有人类最高智力的象棋大师。据说，计算机的应用将像电话和电一样普遍，每座房子里都安装一个控制盘，而且与数字化的“电力工厂”远程连接，由此使家庭成员可以自由出入电子图书馆、自动购物等。这是一个美国式乐观的技术狂热期，一切都集中在计算机“大脑”能够做什么。“到处都是乐观的情绪，觉得计算机能够解决所有的问题，”艾伦回忆道，“这是一个全新的领域，人们还不了解计算机的不足之处。这就是那个时代的一部分。”

分时技术和人工智能是 20 世纪 60 年代最崇高的两个追求，吸引了大量资金和关注。20 世纪 60 年代后期，供普通大众使用计算机、“共享”大型机时间的场所开始在全美各地迅速出现，华尔街也开始向太协计算机分时服务公司（Tymshare）和大学计算公司等注资。分时技术的出现被认为是计算机应用行业的开始。然而，到了 20 世纪 70 年代初，散布全国的分时系统的前景开始变得黯淡无光。人们逐渐意识到，大系统，甚至是可以同时为数百名用户提供服务的系统，越来越难以正常工作。软件再次成为技术瓶颈，运行多用户分时系统所需的编程非常复杂，设计和制造成本超出了所有人的预计。20 世纪 70 年代中期出现了小型廉价的硬件，即台式微型计算机，也就是后来广为人知的个人电脑。这让普通大众接触计算机的梦想再度复活。

20世纪60年代，让计算机赶上人类智力水平似乎成为人们迫不及待要实现的目标。有人预言，计算机将能够看、说、独立解决问题并且从经验中学习。早期的进展的确令人振奋，到了20世纪80年代，乐观的情绪再度高涨，风险投资家嗅到了科技领域下一个大事件的气息，又向人工智能创业公司注入了大量资金。但是，用具有商业价值的方法模仿人类智慧比预想的要困难得多，于是，风险资金又纷纷撤出。软件再次成为拦路虎。

不过，分时技术和人工智能对人们今天使用计算机的方式以及程序员编制软件的方式都产生了影响。分时技术影响了从文字处理器到人机研究的方方面面，最终形成了如今个人电脑上常见的点击图标。人工智能带来了新的编程工具，以及从语音识别到机器人学等计算机应用的新愿景。尽管它们到来的步伐远远慢于人们的期望，但是这个领域的研究在不断地进步，例如，“深蓝”就在1997年击败了国际象棋世界冠军加里·卡斯帕罗夫。

20世纪60年代，分时技术和人工智能的研究都在进步。在这两项研究的起步阶段都起到核心作用的人是约翰·麦卡锡。对分时系统的第一次公开表述是在1959年6月于巴黎召开的一次联合国教科文组织会议上。当时，英国计算机科学家克里斯托弗·斯特雷奇发表了一篇标题为“大型、快速计算机的分时技术”的文章。在此之前，这一概念已经出现了一段时间，自20世纪50年代中期开始，麦卡锡就和同事展开过讨论。“坦白说，我很奇怪，并不是每个人都认为这很容易理解。”[2]斯坦福大学荣誉教授麦卡锡在他的办公室里回忆道。

20世纪50年代，计算机一次可以处理一项工作，或者在打孔卡片上按顺序处理一组任务，即后来所谓的“批处理”。某个人要想使用计算机，通常先将程序写在画好列和行的专用纸上，然后用打孔机把经过编码的指令打到马尼拉纸质卡片上，上面的小孔指示程序员要把数据段放到某个位置上或对数据进行处理。用户满怀虔诚地来到计算机中心，递上这些卡片，却只可以得到有限

配给的宝贵时间。要是某个倒霉的用户编的程序中有 bug，他还得恳求一些额外的时间，也许要等数天之后才能轮到他。

计算机的应用越来越广泛，而等待获得机器时间也让人越来越有挫败感。这个问题首先在麻省理工学院这样的地方突显，因为在 20 世纪 50 年代后期，麻省理工学院是计算机应用方面的领头羊，而麦卡锡当时在此担任助理教授。学院的计算中心率先拥有可操作的分时系统，这是由罗伯特·法诺和费尔南多·考巴托领导的项目。在 1961 年 11 月进行演示时，系统的原型被称为兼容性分时系统，它只有 4 台终端与主架构相连。正如考巴托后来指出的："分时技术的起源是约翰·麦卡锡写的一份关键的备忘录。"[3]1959 年 1 月 1 日，在写给学院计算中心主任菲利普·摩尔斯的备忘录中，麦卡锡主要从编程生产力的角度描述了分时技术[4]，将其作为一种"能够大大减少用机器解决问题所需时间"的方法。他认为这种收益会很大，并特别提及"保守估计效率会是原来的 5 倍"。向来高调的麦卡锡补充道："我认为这个方案指明了未来所有计算机运行的方式，在计算机应用之路上，我们有机会引领一个重大的进步。"

在他的备忘录中，麦卡锡罗列了分时系统所需的技术，包括"中断"特性和快速的内存分配。为了使分时技术发挥作用，计算机必须于同一时间在多个用户需求之间周旋，从一个任务快速转向下一个任务，并能把一个任务"中断"足够长的时间，以便接受其他用户键入的内容。分时技术依托于运行速度更快的新型机器，充分利用人类思考以及通过键盘终端敲入指令的时间，在多个用户之间分配计算周期。

我们可以看出，1959 年 1 月的备忘录是一份有关交互式检错系统提议的精彩文件。详细阐述观点的同时，麦卡锡指出："借助程序员清晰的记忆，写完程序之后立刻检查可以节约更多的时间。"不过，麦卡锡还有更大的想法，他认为分时技术可以带给每一位用户一种错觉，即计算机是他自己独享的，就像个人电脑一样。在美国国防部先进研究计划署的慷慨资助之下，麻省理工学院后来启动了一项名为 Multics 的庞大的分时技术实验。美国国防部先进研究计

划署是 20 世纪 60 年代众多计算机科学项目的赞助人。但是，对这种慢吞吞的进展，麦卡锡逐渐失去耐心。用考巴托的话说，麻省理工学院的领导层认为他是一个“气人而有趣的狂人”，而不是一个可以将其提案作为行动计划对待的人。于是 1962 年，麦卡锡来到斯坦福大学，集中精力研究他最感兴趣的人工智能。

约翰・麦卡锡做过的最气人的一件事是创造了一个新词。1955 年，为了给翌年夏天在达特茅斯大学举办的一个会议寻求资金，他给洛克菲勒基金会写了封信。信中使用了这个词，当时他是达特茅斯大学数学系的助理教授。“我在写给洛克菲勒基金会的申请中杜撰了‘人工智能’这个词。”[5]麦卡锡解释道。虽然听众主要是参加此次会议的人员，但是他希望这些人把眼光放长远一些。根据申请书的陈述，达特茅斯大学的夏季研究项目将“在如下推断的基础上继续推进，即理论上可以精确描述的有关智力学习的各个方面或其他任何特征，都能够用机器来模仿”。麦卡锡回忆道：“我的想法是要树起这面旗帜，事实上它就是一面旗帜。”

用计算机进行智能模拟这个想法最初是在 1948 年 9 月引起了麦卡锡的兴趣，当时他在自己的母校加州理工大学参加了一个主题为“大脑的行为机制”的研讨会。据他回忆，在这次研讨会上提交的论文中，只有几篇对计算机和人类大脑进行了比较。他不无失望地说道：“1948 年 9 月还没有出现存储程序计算机，这在某种程度上也制约了当时所做的比较。”但是，这却促使麦卡锡去思考如何继续深入研究计算机模拟智能的理论。

这一理论可追溯到 1937 年。当时，英国数学家阿兰・图灵描述了一种理论上的计算机，他称为“万能机器”。起初，他进行了这样的论证：只要给予正确的指令以及无限的纸张和时间，书记员就能够解决专业数字家能解决的任何问题。这一论证的含义是，万能机器或者计算机能够模拟包括人类大脑在内的任何计算设备的工作。这正是令无数软件设计师恼火的一种推论。他们认为，

很多事情理论上可行，事实上却完全不可行。据工程学批判文章称，这类荒唐的念头无异于“图灵沥青坑”。但数十年来，图灵的万能机器逻辑推论一直激励着像人工智能研究员这样的人在计算机领域孜孜不倦地探寻。他们会问：既然理论上可行，那为什么不试试呢？

1950 年，图灵在其论文“计算机器与智能”（Computing Nachinery and Intelligence）中直接提出了机器与人类智能这一主题。论文的开头写道：“我提请大家考虑一个问题：机器能思考吗？”[6]这引起人工智能领域研究人员的共鸣，就像美国小说家梅尔维尔《白鲸》中的那句“叫我以实玛利”引起了小说家的共鸣一样。在论文中，图灵首先清晰地表述了关于计算机能否模拟人类智能的测试：向计算机提问。如果人类不能分辨计算机的回答与人类的回答有什么不同，那么就可以认为，计算机已经达到了人类的智力水平。这就是著名的“图灵测试”。图灵的论文接下来描述了“学习机”，它仍然是人工智能的设定目标之一。如果经过适当的编程，计算机就可以从累计计算（即人类所谓的“经验”）中“学习”，从而“改变机器的操作规则”。图灵以一种既富于想象力又谦逊的语句结尾：“我们期待在所有的纯智力领域，机器最终将会与人展开竞争。但从哪个领域开始最好呢？……我们虽然只能看到未来很短的一段路，却明白还有很多的事情要做。”

麦卡锡并未重视所谓的图灵测试。他说，图灵 1950 年的论文主要是与否认机器可能会模拟人类智能的人进行争论。也就是说，如果一个人无法辨别人类和机器的回答，“那么，你所持的异议就是出于偏见或别的什么原因”。[7]麦卡锡补充道，不幸的是，其他人已经把图灵的论断变成了一成不变的人类智力水平测试，因为它并不是一个好的测试。“如果你认为某人不能辨别机器和人的应答，”他说，“那么第一个问题是：谁不能分辨？总有那么一些人，太容易被愚弄。”相反，麦卡锡认为图灵的主要贡献在于他认为“人工智能的关键是计算机程序，他是第一位还算有些正确想法的人……”。

麦卡锡很快就发现，实现上述任务的可用编程工具不多。他解释说，为了

使计算机达到人类的智力水平，我们首先必须“用逻辑的方式重新描述与世界有关的事实”。但在当时，计算机主要用于数字运算。很显然，数字之外还有大量的事实。1958 年，成为麻省理工学院的助理教授不久之后，麦卡锡开始设计一种后来被称为 Lisp 的编程语言。之前他就一直在研究，但是麻省理工学院直到 1958~1962 年间才实施 Lisp，并将其应用在人工智能领域。当时，占据主导地位的高级编程语言是 FORTRAN，但它是为解决科学与工程领域的数字运算问题而量身定做的。不过，人类大脑也擅于处理符号，比如构成自然语言的字母和单词。要想实现人工智能，麦卡锡需要一种能够兼容符号问题和数字问题的编程语言。而且这种语言必须非常灵活，因为只有这样，计算机才能自如操作这些符号，就像人类大脑那样，探索不同的推论、假设以及新的事实。

创造一种灵活的、使用符号的编程语言堪称一个巨大的进步，因为这并不是单纯的理论研究，而是创造出一种能在机器上运行的语言。通过创新工具、技术和灵感的融合，Lisp 实现了这一目标。麦卡锡选用一系列信息列表来构建他的语言，这些信息列表随后将被处理，Lisp 代表“列表处理”（list processing）。各种信息可以显示在列表中，然后用于推论和逻辑推理等操作。当程序运行时，通过自动清除和释放计算机内存，Lisp 将程序员从某些机器内务处理任务中解放出来，这一特性称为“垃圾回收”。

麦卡锡使 Lisp 程序能够重新表现为 Lisp 数据，这在灵活性方面是一个真正的进步，赋予存储程序计算机一种新的能力，即自我引用，允许数据和程序没有任何约束地进行混合。因此，Lisp 的设计引入了“递归”这个过程，即程序或代码段能够调用其自身，来解决那些需要重复处理的问题。此外，Lisp 能处理“深度嵌套”或嵌套循环的编程定位问题，能够使程序跳出循环并且在解决问题之后返回最初的任务。太阳微系统公司的计算机科学家、写过 Lisp 和 FORTRAN 著作的作者盖·斯蒂尔指出：“Lisp 具有 FORTRAN 所欠缺的自由度和灵活性，它能够以某种方式返回到自身，而 FORTRAN 不具备这种机制。”[8]

Lisp 并不是第一门列表处理语言。在 1956 年的达特茅斯会议上，艾伦·纽

威尔和赫伯特·西蒙就描述过一种名为 IPL 2[9]的列表处理语言。他们是人工智能研究方面的先驱，后来都成为了卡耐基梅隆大学的教授。但是，IPL 2 过于依赖兰德公司的 Johnniac 计算机的特性，因此注定不能广泛使用。20 世纪 60 年代，还出现了其他使用符号的编程语言，包括贝尔实验室的 Snobol、麻省理工学院的 Comit，以及 IBM 的 Formac 等。但是，最为领先的是 Lisp，而且它证明了计算机不仅能处理数字，也具备处理符号的基本能力。Lisp 还深深地影响了其他编程语言的设计人员，如 Java 互联网编程语言的创始人詹姆斯·高斯林。

但是，Lisp 从来都没有成为主流的编程语言。不过，它的应用范围很广，其设计初衷不单单是为了商业应用，不像 FORTRAN 主要用于工程和科学研究、COBOL 主要用于一般的商业数据处理、SQL 主要用于数据库那样。Lisp 的标记方法有助于赋予其灵活性，但其填满括号的软件语法这一表现形式令许多程序员不快。初期，它破坏了计算循环，而且运行缓慢，后来当计算机的速度越来越快且价格越来越低时，这种不好的声誉阻碍了 Lisp 的发展。20 世纪 80 年代人工智能商业化的失败也进一步使 Lisp 魅力大减。现在 Lisp 只是小众语言，拥有少量却忠实的追随者。

马文·明斯基与约翰·麦卡锡一起开创了麻省理工学院人工智能这个学科。明斯基、麦卡锡、纽威尔和西蒙四人被称为“人工智能学科之父”。当被问及如何定义人工智能时，明斯基曾这样回答：“如果年轻人打算让机器做他们还不会做以及用现有的方法很难做到的事，那么人工智能实验室就是他们的舞台。所以，从这个意义上说，人工智能就是计算机科学领域更具远见的部分。”[10]虽然科研人员不断取得进展，但是人工智能的前景却愈发黯淡。一种结果就是人工智能不可避免地遇到特定的膨胀问题。即便计算机能做一些事情，人们也根本不认为有什么了不起。计算机击败了国际象棋世界冠军？好吧，还是看看原始计算机语音识别的情况吧。

但是，事实已经证明人工智能远比想象中的还要困难，这也是不争的事实。例如，1958 年，纽威尔和西蒙预言[11]，到 1970 年计算机将能创作出古典乐曲，发现重要的新数学原理，而且还能理解并翻译口语。拉吉·瑞迪是约翰·麦卡锡在斯坦福大学的早期学生，他于 1963 年加入这个研究课题，并成为斯坦福大学首批计算机科学博士学位的获得者。瑞迪如今已是卡耐基梅隆大学的教授，是人工智能领域，尤其是语音识别和机器人学研究领域的领先者。瑞迪回忆，在 20 世纪 60 年代，“我们认为一切皆有可能”。他说：“我们觉得语音、机器人以及所有其他问题都会在未来几年内得到解决。对于计算机达到人类智力水平到底有多难，我们完全没有概念。我们过分低估其间的复杂性了。”[12]

将近 40 年之后，瑞迪这位业界元老指出，人工智能“其实已处于起步阶段的末期”。而体格魁伟、须发灰白、脾气暴躁、已过古稀的麦卡锡，依然是他帮忙创立的这一学科的坚定捍卫者。他说，那些声称人工智能失败的人大多是“无知的记者”和锱铢必较的风险投资家。麦卡锡似乎已将“无知的记者”这个词当成了口头禅。“绝大部分否定人工智能的人都是投资者，”他说，“他们在这上头输得精光，但仍然富裕，还有足够的影响力。”[13]麦卡锡生长在一个激进的工人阶级家庭，父母都是共产主义积极分子，他们给儿子灌输理性主义信仰，坚信技术有益于全人类。没有任何领域能像人工智能一样，如此具体地展示技术带给人们的自信。麦卡锡固然脾气急躁，但始终是个乐观主义者。“达到人类水平的人工智能是一个科学难题，目前还没有解决。”他说道，“谁知道还要多长时间？不过，从孟德尔①到人类解开遗传学密码，毕竟也花了 100 年。还没有人敢说遗传学已经大功告成了，人工智能同样也还没有大功告成。”[14]

到 20 世纪 50 年代末，计算机的应用范围已经超越了科学和工程领域，逐

① Gregor Johann Mendel（1822—1884），奥地利遗传学家，天主教圣职人员，遗传学的奠基人。——译者注

渐参与到那些正处于转变过程中的大公司和政府组织的核心业务中，正如《财富》杂志在 20 世纪 60 年代中期指出的那样，计算机是“近十年内引进的最重要的管理工具”。[15]计算机能够协助管理战后日益复杂的、在规模和范围上不断增长的会计账目、工资支付、物流、采购及生产运营等，这一点已得到普遍认可。毫无疑问，需要针对这些管理和业务问题进行编程简化，正如 FORTRAN 为科学和工程难题所做的简化一样。让委员会制定一种编程语言将是异常艰难的，但是，这是一种相当成功的方式，使该语言成为了商务编程的坚实标准。起初，对于 COBOL 这种面向商务的通用语言，还有一些计算机厂商的零星抵制，但这些抵制很快就被证明是徒劳的。从一开始，COBOL 就不被学术界重视，不过，负面的评论并没有影响人们去接受它。由于 COBOL 主要是用英语编程的，所以它似乎是一种容易学习的高级语言。而且，它解决了那个年代的一个大难题：如何在编程语言中处理业务数据。

说来也怪，COBOL 的成功既说明 20 世纪 60 年代初期计算机正逐步得到人们的认可，又说明这仍是一个很小的行业。计算业务很少而且集中，单个客户就能确保 COBOL 成为一种技术标准。美国国防部是当时购买计算机的最大客户，他们监管了 COBOL 的设计，而且政府在其完成后宣布，除非计算机使用 COBOL 语言，否则它既不会购买也不会租用。（20 世纪 70 年代，国防部曾号召行业支持由政府资助的编程语言 Ada，但是并不成功。）

COBOL 开始的确得到了大多数计算机生产商和政府及行业用户的支持。1959 年 COBOL 面世时，有两家公司已经开始着手建立自己的商用语言：斯派里 · 兰德公司 UNIVAC 部门的 Flow-Matic 是一种已经使用数年的数据处理语言，而 IBM 为 Commercial Translator 开发的 Comtran 才刚刚启动。其他计算机生产商并不愿意看到竞争对手设计出一个重要的新编程标准，因为即便这种语言公开发布、不收取费用，就像 IBM 的 FORTRAN，但对于创建这种语言的公司而言，拥有一种标准的商用编程语言将掌握巨大的市场优势。计算机用户担心的是，在无序的市场竞争催生出新的标准之前，势必要经历长期的混乱。除

斯派里·兰德和 IBM 以外，其他公司也都宣布了他们开发商用语言的计划。谁能率先成为商用语言方面的大赢家，前景并不明朗。

采用集体的方式设计商用语言，最早是由宝来公司（Burroughs）的编程经理玛丽·霍丝提出的，当时宝来公司并没有参与商用语言的角逐。当她把这个想法告诉宾夕法尼亚大学的索尔·考恩教授时，他们一致认为，为了促进这个年轻行业的发展，应该开发一种商用语言。1959 年 4 月，在宾夕法尼亚大学计算中心召开了一次会议，与会人员只有少量的客户及生产商代表，其中包括葛丽丝·霍普，她负责斯派里·兰德公司的 Flow-Matic 的开发。在这次会议上，霍普建议由美国国防部担任这项集体事业的总指挥。要达成切实可行的协议，政府要发挥必要的领导作用和影响力。

于是小组成员去见了国防部数据系统研究主任查尔斯·菲利普斯，后来他同意出任 Codasyl（Committee on Data Systems Language，数据系统语言委员会）的主席。5 月底，在五角大楼召开了为期两天的会议，大约有 40 位计算机厂商和用户代表参加。这次大会达成了一些框架性协议，特别是“最大限度地使用简单英语”[16]，菲利普斯在这次会议上的一份报告中写道：“我们需要广泛团结那些能够表述计算机问题的人。”不久之后，Codasyl 执行委员会召开了小型会议，与会人员包括美国钢铁、杜邦以及埃索等公司用户的执行委员，以及两位计算机行业的顾问：斯派里·兰德公司的霍普和 IBM 的罗伯特·贝莫。该机构将对实际设计 COBOL 语言和从事细化工作的团体进行审查和监控。该团体最初是由六男三女组成的“短期委员会”，于 1959 年 6 月的最后一周召开首次会议，并于当年年末交付 COBOL 语言。在 6 月的首次会议结束之后，又有其他人加入了这个短期委员会。

短期委员会由像珍·萨梅特这样经验丰富的编程人员构成，她从 4 年前就开始从事编程工作，是当时知名的资深人士。和大多数早期的编程人员一样，

萨梅特步入这一领域实属偶然。她的父母都是纽约的律师，萨梅特甚至在认识数学这个词之前就表现出了数学方面的天赋。她在曼荷莲（Mt. Holyoke）学院主修数学，随后获得了伊利诺伊大学的硕士学位。正是在伊利诺伊大学，她才对学校的 Illiac 计算机有了些了解，但她一点儿也不想跟计算机扯上关系。对她而言，数学是高尚的知识探索，而工程是机械师的肮脏工作。这是那些从事数学理论学习的人相当普遍的看法。“很难描述我们对计算中心的那些工程师有多么藐视，”萨梅特回忆道，“在我看来，计算机就是一些脏兮兮的硬件，我可不想碰这些东西。”[17]

回到纽约之后，萨梅特得到了一份哥伦比亚大学助教的工作，同时她开始在那里攻读博士学位。学年开始前，萨梅特在大都会人寿保险公司的财务部门找到了一份工作，负责红利计算。大都会人寿当时正使用大型计算机，正是在这里，萨梅特参加了一个为打孔机计算做准备的培训课程。她说：“让我大吃一惊的是，我竟然喜欢它。”离开哥伦比亚大学之后，萨梅特前往长岛的斯派里陀螺仪公司工作，当时正值美国国防部决定要建造一台计算机，于是，她成了这个团队的一员。萨梅特回忆，1955 年年初，她的老板亚瑟·豪泽问她是否愿意成为一名程序员。萨梅特回答说：“什么是程序员？”豪泽回答说他也不清楚，他只知道那个计算机项目需要一名程序员。萨梅特接着问是不是有点像摆弄打孔卡设备之类的，老板说可能是。“就这样，我成了一名程序员。”萨梅特笑着回忆这段往事。

斯派里公司早期研发的计算机称为 Speedac，主要供公司和相关人员学习体验之用，当然也包括程序员在内。这家公司明白，必须接受数字计算机，哪怕仅仅是因为它们将要成为国防工业中至关重要的工具。1955 年，斯派里公司收购了雷明顿·兰德公司，从而进入了计算机行业并发展得更好，而兰德公司凭借其研发的 UNIVAC 机器成为了商业计算领域早期的领先者。对萨梅特而言，这次收购意味着她要做好在强大的 UNIVAC 计算机上工作的准备。此后，她和几位同事将定期乘火车前往费城的 UNIVAC 部门，参加晚上的培训，这样

她就可以在新机器装船运给客户之前，利用晚上的时间在上面测试程序。“我们当时是 β 测试人员，实际上做的是 α 测试人员的工作。”萨梅特这样说道。当时这种做法得到了上级 UNIVAC 经理葛丽丝·霍普的鼓励。

此外，萨梅特还在阿德菲大学的夜校教授编程。她的教学生涯始于 1956 年秋天，需要在没有教科书的情况下完成了第一学年的教学工作。在第二个学年开始之前，丹·麦克拉肯已经出版首部关于编程的书籍，于是她就用这本书来授课。1957 年，FORTRAN 发布，她就开始一边教这种新的编程语言，一边匆忙地阅读 IBM 的用户手册，只比她的学生领先一两步。“我对 FORTRAN 的了解一点也不比对盆栽植物的了解多。”在马里兰州贝塞斯达的公寓楼内，萨梅特指着会客室的观赏植物说。这种直言不讳的说话方式贯穿萨梅特职业生涯的始终，而她也一直从事着兼职的教学工作，后来创建 C 语言的丹尼斯·里奇，就是她在波士顿授课时教过的一名学生。

多年来，萨梅特在课堂上及工作中培养了大量程序员。她发现，预测一个人能否成为一名优秀的程序员并不是件容易的事。凭借对精确性、逻辑和数学的喜爱，萨梅特掌握了一些诀窍。她指出，数学能力可以作为一个很好的指标，但决不可一概而论。她回忆起同时教的两个程序员，一个是年轻姑娘，拥有家政学学位；另一个是小伙子，在应用数学系读研究生。据她回忆，学家政学的这位是个“极佳的程序员”，而那位学数学的则是个废物。“一个人是否具备某些心理特征决定了他能否成为优秀的程序员，这与教育背景基本上没什么关系。”萨梅特说道。在招聘员工时，她总是寻找那些具备某种知识热情、愿意沉醉于代码和机器并从中获得最大乐趣的人。萨梅特于 1958 年加入 Sylvania 电气产品公司，当时她总是问求职者，他们是对执行业务和科学任务的应用软件等“有用的”编程感兴趣，还是对开发编译器、设备控制器以及用于计算机内部结构控制的其他软件等“无用的”系统编程更感兴趣。“如果答案是‘有用的’，我就告诉他们‘我不用你’。”她说着，回忆起自己半开玩笑似的回答。

萨梅特被派往 COBOL 短期委员会，主要任务是“针对数字计算机编程，推荐一个短期（至少在一两年之内有用）的通用商务语言的综合办法”[18]。这种“综合办法”的关键是要综合现有的 3 种商用语言项目：斯派里·兰德公司的 Flow-Matic，IBM 的商用转换器，以及主要借鉴 Flow-Matic、由空军主导的 Aimaco。这个短期委员会的通用商务语言将独立于机器，也就是说它是一种更高级的编程语言，各计算机制造商据此编写针对特定机器的编译器。另外，还有一个中期委员会以及计划中的长期委员会。事实上，中期委员会从未制定出一种语言，而长期委员会则根本没有成立。短期委员会称自己是快得要死（PDQ，Pretty Damn Quick）委员会，甚至在其书面报告中也这样说。萨梅特后来写道：“我们大多数人都将自己的工作成果视为权宜之计——它的确是相当重要的权宜之计，而非什么一劳永逸的东西。”

最初给委员会设定的工作期限是 3 个月，但很快就发现这不切实际。他们要花近 6 个月的时间，到 1959 年 12 月才能完成任务。除了技术规范之外，还有为这种新语言命名的问题。在 9 月 17 日的一次会议上，他们讨论了如何命名，共提出了 6 个名字，包括 Busy（Business System，商务系统）、Infosyl（Information System Language，信息系统语言）、Datasyl（Data System Language，数据系统语言）以及 Cocosyl（Common Computer System Language，通用计算机系统语言）等。看起来没有一个恰如其分，最终也没有达成共识。葛丽丝·霍普和罗伯特·贝莫虽然不是上述短期委员会的成员，但作为监督 Codasyl 执行委员会的顾问，他们对这项工作保持着密切的关注。那天晚上，霍普和贝莫也讨论了命名问题。“谁都提不出完全名副其实的名字。”贝莫回忆说。[19]于是，身为字母缩略法的创始人和 Codasyl 的倡议者，贝莫提议叫 COBOL。他回忆道：“葛丽丝说：‘行，我听着还不错。’也没有什么真正意义上的讨论。”第二

天，作为面向商务的通用语言（Common Business Oriented Language）的简单缩写，COBOL 获得了认可。这一看上去并没有蕴涵更多含义的名字注定要成为一个难以磨灭的计算术语。（事实上，贝莫在 1971 年发表的对编程语言的评论中写道："我们不认为是某个个人创造了'COBOL'这个缩写。"[20]后来，霍普把想出这个名字的功劳归于贝莫，霍普于 1992 年去世。贝莫听到霍普讲述这段轶事的时候，说这也唤醒了他的记忆。得知贝莫的叙述之后，萨梅特说她对此表示怀疑。）

到 1959 年 10 月，短期团队已经取得了一些进展，但是他们却陷入了无休止的争论之中。除了最初的 9 名成员，委员会还吸纳了其他一些行业的代表。通常有 12~20 个人参加会议。这些会议就像一个论坛，毫无效率可言，却试图为一种编程语言制定出技术规范。"实际上，委员会所能做的很有限，"萨梅特回忆道，"人们只是围坐在那里高谈阔论，不会有什么结果。"后来他们决定从委员会成员中选出一个 6 人小组，负责制定这种语言的详细规范，其中就包括萨梅特。这个 6 人小组是 COBOL 的实际创建人。他们的大部分工作是在纽约完成的，他们连续两个星期都待在纽约雪梨–荷兰酒店里不分昼夜地忙碌，直至最终结束。11 月初，小组提交了报告初稿。随后的 5 天，短期委员会对报告初稿进行了审阅和讨论。虽然有一些修改建议，但是他们绝大部分的工作成果都得以保留。

作为一种编程语言，COBOL 的模样怪怪的。如果使用 COBOL 包含的相当大的英文字符集，能够写出看起来像是自然语言的表述：

```
MULTIPLY HOURLY RATE BY HOURS WORKED GIVING GROSS-PAY.
```

编程语言的纯化论者常常觉得 COBOL 给计算机界抹了黑。不过并没有编程语言方面的理论家参与 COBOL 的设计。相反，COBOL 的开发者是由商业编程人员组成的小团队，在极其严苛的时间压力下完成的。他们的主要任务是从一些数据系统语言的早期实践中选取材料并将其糅合在一起，同时尽可能多

地使用英语。这么做的初衷是让那些非专业人士也能够轻松地进行编程，让管理层感觉商务计算也是可以理解的，并非那么令人生畏。然而，对于商用编程语言到底应该简单到何种程度，即便是 COBOL 语言开发团队内部，认识上也存在很大分歧。

争论的焦点在于“算术动词”问题。一方认为不应该强迫商务用户纠结于数学公式，所以这种语言就应该包括加、减、乘、除（ADD、SUBTRACT、MULTIPLY、DIVIDE）等算术动词；而另一方认为，上述算术动词是语言中愚蠢和弱智的内容，而且任何使用计算机工作的人都熟悉简单的数学公式。争论最后以妥协告终：COBOL 语言中既加入算数动词，也允许出现公式。不过，在公式表达式之前，必须加上动词 COMPUTE。“最大的争议在于算术动词，”萨梅特说，“因为这个问题使派系的分裂暴露了出来。”

在数据描述和表达方面，COBOL 的确给编程领域带来了重大进步，这反过来又为数据库技术的发展做出了贡献。例如，COBOL 包含称为“图像子句”的特征，能够用分级表的方式描述数据。这种应用方式可以帮助组织程序中诸如姓名、地址、社会保障号以及电话号码等基本的业务或客户数据等。程序员采用一组定义的数字或字符，以可视化的模式创建数据的“图像”。例如，可以用数字 9 代表“图像”中同样位置对应的数字。这样，存储在计算机中的电话号码的初始数据可能是：2125561234。通过图像子句——(999) 999-9999——过滤之后，显示在用户屏幕上就成了(212) 556-1234。“COBOL 在分级数据布局方面做得非常好，而用看待 FORTRAN 和 Algol 的眼光来看的话，这根本不值一提。”普林斯顿大学的布莱恩・科尼汉教授这样评价说。[21]

COBOL 并没有完全实现其暗示的“用英语编程”的承诺。这一概念无疑很吸引人，而葛丽丝・霍普正是其最大的拥护者和推动者。她后来成为软件界举足轻重的领军人物，还获得了海军少将军衔。霍普身材瘦小，精力旺盛，说

话非常坦诚，是一位技术传道士，她提出软件要更加广泛地普及，这一愿景推动了整个行业朝着这个方向发展。她用讲故事和奇闻轶事的方式来解说这项深奥的技术，让计算领域之外的人也能够理解。事实上，她非常习惯于这种简单化的夸大其词。

她颇具编故事的天分，例如把计算机术语 bug 永远跟她联系在了一起。她不厌其烦地讲述一件轶事，说在 1947 年 9 月 9 日，她用镊子把一只死蛾子从 Harvard Mark II 计算机里夹了出来。[22]“从那以后，”霍普在 1981 年的一次演讲中说，“每当计算机出了什么毛病，我们总是说里面有 bug。”然而，bug 这个词用来指机械方面的小故障至少可以追溯到 19 世纪，爱迪生在 1889 年就说过在他的留声机中发现了 bug。而在史密森尼学会的美国国家历史博物馆中保存的霍普的论文表明，在 1947 年之前好几年，霍普和其他人就已经用 bug 这个词来描述计算机问题了。但蛾子的故事仍在流传。

霍普是 COBOL 项目背后的主要推动者。在斯派里·兰德公司的时候，霍普领导开发的 Flow-Matic 后来对 COBOL 的贡献很大。COBOL 的开发受 Codasyl 执行委员会的监督，她是该委员会的顾问。为了说服各地的公司采用 COBOL，霍普在全国做巡回演说。她作为开发者、发明者或“COBOL 之母”而广为人知，事实上，霍普并没有直接从事 COBOL 的设计和技术规范的制定。“有关 COBOL 的最错误说法就是说霍普发明了它，”萨梅特说，“我所参加的任何一次委员会会议都没见过她……我比任何人都敬佩霍普，但她并不是开发者、发明者或 COBOL 之母。”

当被问及她在 COBOL 的工作中所扮演的角色时，霍普非常坦率，甚至非常明确。1976 年，计算机书籍作家丹·麦克拉肯告诉霍普，他打算将自己的《COBOL 简要编程指南》一书献给她，并欲称她为“COBOL 之母”。霍普回答说“不”，因为 COBOL 继承自 Flow-Matic，所以更确切地说，她应被称为“COBOL 之祖母”。麦克拉肯指出：“葛丽丝用自己的方式表示了谦虚。”[23]

然而她是个出色的推销员。麦克拉肯回忆起在费城参加霍普的一次有关

COBOL 的讲座。她着重强调了英语编程，指出只要保持 COBOL 有限的英语词汇并对问题有选择地分类，它就能很好地发挥作用。这是一种误导，麦克拉肯说，展示 COBOL 可允许人们用英语编程会引起误会，让人们误以为 COBOL 是一种具备自然语言灵活性的编程语言，能够适应人类表达的模糊性和不精确性。但是就像汇编语言一样，一个不恰当的句号，或者一些其他的错误输入，都可能使 COBOL 程序崩溃。麦克拉肯指出："COBOL 就像其他任何编程语言一样挑剔和无情。"

企业管理人员从来没有真正打算去读一读其技术团队编写的 COBOL 代码。专业的程序员很快在自己的代码中采用了助记缩写，以避免像写句子一样写冗长的英语单词。"关于如何在编程中使用英语，葛丽丝被误导了，"麦克拉肯说，"她所期待的方式恰恰没有出现。"他紧接着说，但是在简化编程的必要性以及对商务数据处理语言的需求等大的构想方面，她是正确的。麦克拉肯将霍普与哥伦布做了宽泛的比较，哥伦布驾船向西航行，没有发现香料群岛，却发现了新大陆——如果不提最初目标的话，的确意义非凡。"COBOL 就是商用的 FORTRAN，"他说，"这是一个巨大的成就，而葛丽丝 · 霍普是领路人。"

人们认识到计算机正在企业、商业以及政府运营中发挥重要作用，软件对于这一切至关重要，但也日益成为一个问题。这推动了 COBOL 的发展，也推动了"软件工程"的兴起。IBM 似乎是最适合解决这一问题的公司。负责处理这个问题的是 T. 文森特 · 里尔森，他是仅次于小托马斯 · 沃森的第二号人物。里尔森身高 1.98 米，瘦瘦高高，仪表堂堂，他是沃森的战地指挥官，也是个脾气暴躁的实干家。IBM 的研究人员和技术团队认为里尔森是个聪明绝顶的推销员和执行官——一个与众不同，却还不错的家伙。1960 年，里尔森任命 FORTRAN 团队经验丰富的初始成员戴维 · 赛尔担任编程负责人。赛尔组织了一次 IBM 程序员大会，为公司的软件发展出谋划策。

1961 年 6 月，100 多名 IBM 程序员——按 Sayre 的话说是“现有的全部精英”[24]——集中在新罕布什尔秃头峰的移民俱乐部，召开了为期一周的会议。但是对程序员来说，秃头峰的集会不仅仅是讨论的论坛。沃森、里尔森以及销售和产品部门的领导也全部到会。会议产生了一个“行动计划”，包括将要采取的 35 个步骤。赛尔指出，这些步骤全都来自“认识到 IBM 投放到市场上的产品一半是软件一半是硬件”以及所提出的薪金和软件测试等方面的建议。从理论上讲，软件被赋予了同硬件平等的地位。这样，从表面上来看，似乎意味着以同样的方式对待硬件和软件，甚至使用相似的术语。例如，IBM 已经建立了硬件设计和生产的工程流程。分为 3 个清晰描述的步骤，A、B 和 C，从概念设计到整机，乃至全部测试完毕，准备送交客户。IBM 的工程师为软件测试采用了相同的术语 α、β 和 γ。事实上，直到今天，α 和 β 作为业内的术语仍在广泛应用，不过它们主要用于描述在程序提交用户之前，协助进行无休止的错误排查工作。

从 20 世纪 60 年代开始，人们就期望像对待硬件那样对待软件。平等对待是一种激励，至少一开始是这样，但同时也是为了努力构建更具结构化规范的编程，就像硬件工程那样。回首往事，赛尔指出，秃头峰会议是 IBM“僵化的”编程管理思想的发端。他回忆说，早期 IBM 的编程颇具非正式的硅谷特色，倾向于鼓励创新。到 20 世纪 60 年代后期，工作环境开始受到严格管制。但硬件工程的规范并不太适合更加自由的软件领域。“软件比硬件更具可塑性，”赛尔说，“把它捏圆了再压扁，任凭你自由发挥。”

即便在 20 世纪 60 年代构建大型软件程序面临着诸多问题，力图将软件开发变为一个更加规范、更加工程化的过程仍然是顺理成章、不可避免的。存储和电路技术的进步使计算机变得更快速、更强大。内存和处理速度合起来每增加 10 倍，系统性能就会提高 100 倍，这在 20 世纪 60 年代的前 5 年就已经实现了。然而，机器突然之间取得突飞猛进的发展，那么软件方面的进步就显得不如人意了。试图将程序从 1000 行代码提高到 100 万行代码，直接导致了软

件危机。复杂性的增加主要是生物性的而非机械性的——所有的代码都要以某种方式相互配合、相互适应。然而为大型任务增加人员的做法，通常只会加剧混乱，并没有降低工作强度。软件的构建并不像建设一座桥梁或公路等有形事物那样遵循同比例原则。大型软件项目总是延期交付、预算超支，而且这样的程序通常可靠性较差。

20 世纪 60 年代具有标志性的软件项目，是 IBM 的 360 产品线大型计算机操作系统。OS/360 是一个大胆的、野心勃勃的设计，其中既包含有聪明的点子，也有糟糕的主意，简直是一个软件大杂烩。操作系统常常被描述为计算机的命令和控制系统。如今，最著名的操作系统是微软的 Windows、苹果的 Macintosh 以及 Unix 操作系统，其版本从 Sun Microsystem 的 Solaris 到自由共享的 Linux。到 20 世纪 60 年代初，用户使用计算机完成的任务范围之广泛前所未有。为了满足客户的需要，大多数计算机生产商都提供基本的操作系统。客户希望自己的程序员更加关注科技和业务问题，而不是机器的内部运行机制。因此，由生产商提供的操作系统通过机器将工作流程协调统一起来。

OS/360 是对 20 世纪 60 年代初期操作系统概念的大胆扩展。虽然它曾差点儿因为不合格而成为 IBM 的灾难，但最终凭借其自身的奇妙创意、IBM 的雄厚财力以及编程队伍破釜沉舟的艰苦努力而幸免于难。对 IBM 来说，360 大型机产品线是一个巨大的成功，它确保了公司在行业内的支配地位一直持续到 20 世纪 80 年代。而且，OS/360 成为了行业标准技术，是那个时代的 Windows。直到现在，世界上许多有实力的企业和政府部门的运算都是由大型计算机完成的。运行的软件虽然更新了一代又一代，但都算是 OS/360 的直系后裔。

1964 年 4 月发布的 360 产品线包括 6 种机型。360 这个名称是指 360 度、全方位计算。构思该产品线时，考虑到了市场营销和客户的软件。该产品线不同型号的计算机可以“兼容”，意思是为一种机型编写的程序可以在产品线内的其他机型上运行。这样，随着公司的发展，当客户升级到 360 系列的较大机型时，在科研、会计以及制造软件应用方面的投资将会受到保护。360 包含“仿

真”软件，这样，为 IBM 早期机型编写的软件能够转换到 360 机器。公司还承诺，OS/360 具备“多种编程”能力，能够使计算机同时运行多个程序。这一点具有极强的市场号召力，但履行这一承诺的过程并不轻松，且代价高昂。

1966 年，在同客户的谈话中，沃森用黑色幽默的方式承认了 360 软件带来的痛苦，指出每年用于编程的预算增长得太快了。“几个月前说 1966 年的费用将达到 4000 万美元。”[25]沃森告诉他们，“昨天晚上，我问里尔森，他说要达到 5000 万美元。24 小时后，我在大厅里碰见了瓦茨・汉弗雷，他负责产品编程，我问他‘这个数字没问题吧，我能不能拿来用’，他说费用将会达到 6000 万美元。你们看，要是我就这样问下去，今年我们可就付不起红利了。”

1966 年 1 月，瓦茨・汉弗雷接任 IBM 编程主任一职。1965 年年底，包括汉弗雷在内的少部分人，被要求提交详细的报告，探讨如何更好地管理 IBM 的编程团队。汉弗雷回忆起他的报告是“详论对于规范的计划、进度及跟踪的需求”。[26]IBM 高层正致力于解决 OS/360 生产过程中的延误与混乱问题，这份报告正中下怀。几天之后，里尔森告诉年仅 39 岁的汉弗雷，让他负责公司的编程业务。

汉弗雷自幼就非常熟悉用于克服学习障碍的基本权利和原则。他的祖父曾是纽约联邦储备银行的首任行长，而他的父亲瓦茨・汉弗雷二世，是纽约一家大型保险公司的财务主管。[27]然而，小汉弗雷读一年级的时候就因成绩太差而被退学。那是 20 世纪 30 年代初期，很多有阅读障碍的孩子只是被认为有些“迟钝”。然而，汉弗雷的父母却带着他去检查，让专家做出诊断。为了解决他的问题，全家搬到了康涅狄格州，以便汉弗雷能够进入专门为有特殊阅读障碍的孩子新开办的 Forman 私立学校，在那里他有专门的指导老师，直到他 9 岁。

在学会了应对阅读障碍之后，汉弗雷进步很快，16 岁就高中毕业了。他尤其擅长算术和科学，直到今天他仍然对老师约翰・亚尼尔心存感激，他教的课

程是那么生动（“他改变了我的生活”）。汉弗雷获得了加利福尼亚理工大学的新生奖学金，却没有选择去那里，而是在 1944 年加入海军服役。战后，汉弗雷进入芝加哥大学，主攻物理专业。此后，他在芝加哥大学获得了 MBA 学位，同时还在伊利诺伊理工大学学习了电子工程课程。

1953 年，汉弗雷加入 Sylvania 电动产品公司，负责在波士顿的一个工程小组，为政府部门制造一种特殊用途的密码计算机。为了提升技能，汉弗雷参加了麻省理工学院的一个计算机编程夏季培训课程。该课程由制造首台程序存储计算机的莫里斯·威尔克斯及其在剑桥大学的几位同事任教。汉弗雷的入门编程是编写一个简单的演示程序，在屏幕上产生一个反弹的数字“球”。他回忆起这个程序，虽然编程粗糙，却能够指挥机器执行哪怕是最基本的任务，也是一件令人兴奋的事。他用的机器是国防部资助的 Whirlwind，在当时算是很快的计算机，处理速度达到百万赫兹（2001 年春季的廉价个人电脑的速度是它的 800 倍）。

1959 年，经父亲的一位朋友安排，汉弗雷在与托马斯·沃森本人会面之后加入了 IBM。汉弗雷先后管理了多个工程项目，项目的规模以及对公司的重要性不断提高，直到 1966 年，接手 OS/360 这个烂摊子。他在这个位置上的第一周是清醒的。市场部门刚刚推出了 Blue Letter，用于在时间、价格及编程方面支持 IBM 360 产品线中最大的 91 机型。由于市场部门向客户和业界透露了信息，所以 Blue Letter 在送交销售团队之后，很快就公开发布了。汉弗雷来到完成 OS/360 大部分编程的波基普西实验室，询问了 91 机型的进展。令汉弗雷感到吃惊的是，他发现“这个项目不但没有计划，甚至连人员配备和资金都不到位”。他立即打电话给市场部副总裁杰克·罗杰斯，他还记得谈话是以玩笑开始的：“杰克，我不知道你是个程序员。”他要求推迟发布。91 机型必须等一等，因为还没人为它编写软件。

接下来汉弗雷去了公司的软件开发实验室，却发现他们“接近慌乱的运行

状况”。他们的进度大大落后于时间表，汉弗雷回忆道：“除了最紧急的部分，一切都延误了。”从长期来看，他们所采取的方法只会让事情变得更糟，根本就没有计划可言。当时，软件的交付期限已经推后了 3 次。OS/360 的工作比计划延误了大约一年时间，看起来一切已经到了失控的边缘，必须断然采取行动，而汉弗雷正是这样做的。他跑去找负责开发和生产的高级副总裁弗兰克·卡利，告诉他：“既然所有的交付日期已经毫无意义，我觉得不如干脆全部取消。”然后，他给所有的软件经理 60 天时间，让他们提交各自项目的详细开发计划。拿到计划之后，汉弗雷又为每个项目增加了 90 天时间。

在一个堪称管理效率模范的公司，事情如何到了这样难以控制的地步？如果仅仅因为这是一个技术延伸项目，那么360将是个让人气馁的难题。正如IBM的一位高管对《财富》杂志的一位作者所言：“我们试图为发明制定时间表，对于一个承诺要完成的项目而言，这样做非常危险。”事后看来，这样做的结果是设计可能会被简化，去掉一些多余的或者重复的特点和功能。然而该项目太大了，小弗雷德里克·布鲁克斯说：“当攻坚进入白热化的时候，要想在脑子里分辨出哪些部分重复实在是太困难了。”[28]他曾领导这个项目，不过 1965 年他离开了 IBM，受邀担任北卡罗来纳大学新设的计算机科学系的系主任。

这些困难是由软件设计和开发的特殊难题共同造成的。在布鲁克斯撰写的《人月神话》一书中，他以非凡的思想和智慧，记述了 OS/360 这一传奇故事的惨痛教训。在这部 1975 年出版的关于软件构建的经典著作中，布鲁克斯针对“第二次开发”[29]提出了警告——工程师有一种可以理解却非常危险的倾向，那就是在第二次开发的时候放弃自律。“工程师的首个作品通常中规中矩，没有什么毛病。”他写道，“因为他知道自己并不十分清楚所做的事，所以会非常小心，严格地自我约束。”一旦这种约束获得了成功，在创作第二个作品的时候，就会有些飘飘然。布鲁克斯写道：“因此，人们设计的第二个系统是最危险的。”他同时指出，OS/360 对于大多数设计人员来说，正是第二个系统。

诚然，二次开发综合征也适用于建造桥梁、房屋以及家具等事物。布鲁克

斯的真正贡献在于总结出一些原则，不仅适用于软件，而且是可以广泛应用到计算机程序这样复杂的动态系统的设计和构建中。这些原则中最广为人知的是布鲁克斯法则，即“为延误的软件项目增加人手只会让其继续延误下去”。他说，实际上软件并不像建造棒球馆或历史纪念碑等巨型建筑那样，单凭增加人手就能加快进度。不，软件完全不同。

为了进一步解释，布鲁克斯列举了自己在 OS/360 项目中的艰苦经历，以及当时日益扩大的软件工程领域中的其他人的工作。他说，问题在于，研究表明某些程序员的生产力是其他同行的 10 倍。（某些计算机从业者还争辩说，普通程序员与杰出程序员之间的差距甚至超过 100 倍。）无论倍数是多少，所表达的含义是相同的：编程是一个兼具技巧性和创造性的职业，从事这项工作的人是不可互换的劳动单元。

这样，问题就变成了如何以最高效的方式组织一个大型的软件项目。布鲁克斯警告说这并不容易，但是他确信最佳答案是采用“外科手术团队”的形式。他指出，这个概念是从哈兰・米尔斯那里借用的，他是前 IBM 的经理兼计算机科学家，在软件生产力方面著述颇丰。布鲁克斯解释说，软件项目最好以集中的、专门小组的形式来组织，每个小组都应该“像个外科手术团队，不能像个杀猪队。也就是说，不是让每个人都拿刀解决问题，而是有一个人操刀，其他人全力以赴协助他，这样才能提高效率和生产率”。这样，在分层团队结构中有一个“主任程序员”，而其他各层级人员（如“另一个自我”、测试人员、编辑人员以及秘书等）负责协助他。这样做的目的是让效率最高的人——主任程序员的效率最大化。布鲁克斯说，外科手术团队的结构为主任程序员提供了最好的保障，使其专注于大的、关键性的设计问题，而将实施和管理等事务交给其他人。在大型项目中，时间、精力十分宝贵，错误的“成本”是相当高的，“外科手术团队”式的组织方式能极大地简化沟通，因为关键的沟通发生在主管之间，其他任何人都不参与。

通过取消 1966 年的全部交付计划，并命令 OS/360 各编程小组拟定可靠的

计划，汉弗雷让 IBM 的软件经理们真正负起了责任，不过最终也给予了他们发言权。机器设计在项目中是首要的，因为 IBM 首先是一个硬件厂商。软件被认为是机器的“编程支持”，这方面目光短浅也不足为奇。IBM 不单独出售软件，直到 1968 年，在政府的施压和市场力量的推动下，公司才开始对软件和硬件“分开计价”。IBM 的高级管理层并不真正理解编程，常常根据市场计划来设定编程的进度。“大量问题都来自管理，”约翰・帕尔默指出，他曾是 IBM 的编程经理以及 360 官方历史的共同撰稿人，“在进度设定方面程序员应该有更大的自由。”[30]

计划拿到手之后，汉弗雷宣布了一系列 OS/360 时间表，总跨度为两年。公司逐个按期完成了。虽然晚了一年，并且支出了 4 倍的预算——约 5 亿美元，最初预算是 1.25 亿美元，但 IBM 最终还是兑现了承诺。[31]不过，还存在一些可靠性和性能方面的问题，因此，1968 年，各编程小组仍然承受着巨大的压力，从事紧急的错误排查和维护工作。通常，如何评价一位执行者的影响力并没有确定的标准，但事实上 OS/360 的交付一直严格遵循汉弗雷的时间表。

诚然，正是由于 360 设计人员作出的精明决定，汉弗雷才有足够的时间来处理 IBM 软件的问题。他们植入了特殊的编程电路——“仿真器”技术，允许用户在 360 机器上运行为 IBM 早期的 7000 和 1400 系列机器开发的软件。只要能够运行现有的应用软件，客户就会购买 360 计算机，并愿意为 OS/360 等待，而不是转而购买其他厂商的计算机。仿真器技术为 IBM 赢得了所需的时间，并最终等到了 OS/360 问世。汉弗雷说：“正是它挽救了我们。”

IBM 360 成了第二次世界大战后最经典的商业成功案例，它“押上整个公司”做赌注，并获得了巨大的回报。360 产品线及其后续的 370 机型确立了大型机时代。而进行 OS/360 开发的那几年（1963~1968 年）对于 IBM 的程序员来说，是备受煎熬的时期。“每个小时都是那么漫长，管理层还不断地给他们施加压力。”IBM 史学家帕尔默指出，“很多程序员认为他们把最好的时光献给了 360，却没有得到什么结果。他们心灰意冷，不少人因此而离开了。”[32]

20 世纪 60 年代，高效可靠地构建大型软件项目无比艰难。IBM 360 的软件困境就是确凿的证据。在大型程序中，不可预知的问题接踵而至，有些甚至带来了可怕的后果。例如，1962 年准备发射到金星的“美国水手 1 号”宇宙飞船，在离开发射台不久就失控坠毁了。调查发现，导致问题发生的原因仅仅是火箭导航系统一部分的 FORTRAN 程序中有一个错误字符。

1968 年，北约赞助召开了一次会议，讨论现存的“软件危机”给西方经济安全和军事带来的威胁。很多人在会议上发表了论文和演讲，不过最主要的是，大家几乎一致认同未来还有很长的路要走。没有一个直截了当的答案或“突破点”能够解决软件工程问题。“在软件领域的确存在‘危机’。”密歇根大学的计算机科学家伯纳德·加勒后来回忆道，“每个人都明白，硬件正在快速向前发展，我们也能预见计算机的应用潜能，但却不知道如何用软件开发人员的‘工匠心态’来开发新技术……如何管理一个数百人的团队，如果每个人都要开发出一段完美的代码，并且要同其他人开发的类似代码互相衔接？”[33]

瓦茨·汉弗雷从 20 世纪 60 年代在 IBM 工作的经历中得知，通过合理地应用工程规范，能够将软件做得更好、更可靠，此后他也对此深信不疑。1986 年，他加入了卡耐基梅隆大学软件工程研究所，从此开始长期致力于提高软件的质量。2000 年，印度以他的名字命名了一个研究院——瓦茨·汉弗雷软件质量学院。目前印度正试图凭借高质量在软件领域赢得竞争优势，20 世纪 80 年代日本的汽车行业也是这样做的。对于汉弗雷而言，软件中的麻烦始于其自身的工作。运气不好的编程人员，需要对人类问题进行模糊处理，并“进行绝对精确的描述，以便在机器上运行，这与人类行为不同”。

然而，即便如此，构建几乎没有缺陷的高质量软件仍然是一场旷日持久的斗争。汉弗雷解释说，糟糕的软件可能每 1000 行代码就有 10 个缺陷。也许它能工作，但用户可能会遇到麻烦。如今，大多数商用软件要好一些，每 1000

行可能有 5 个缺陷，不过按照汉弗雷的说法，还是不够好。他接着说，做一个合理的假设，每行代码有 17 个字符，那么 1000 行总共有 17 000 个字符。因此，“糟糕”的代码是指在 17 000 次击键中有 10 次失误。汉弗雷说：“对于软件来说这很糟糕，但是在人类尝试的其他大多数领域，这已经很了不起了。编程是纪律性非常强的一项工作。”他还说，软件领域可以分为两个方面：一方面是技艺创新的大爆发，他称之为“不朽的创造”；而另一方面是坚持不懈的工程工作，包括软件的改善、维护和测试等。“我们所从事的 90%~95%都是工程工作，而非不朽的创新工作。”他说。

确切的百分比可能有所出入，但汉弗雷的观点无疑是正确的。当然，这个观点还不完整，创新可能只占 5%或 10%，但多年来开创性的创新活动为软件领域奠定了基础，并推动其向前发展。如果没有这 5%~10%的真正创新，还用得着为工程过程烦心吗？而每个人在软件中所使用的最好的工具——编程语言和其他工具——就是那些不朽的创造者的工作成果。1993 年，软件工程大师弗雷德・布鲁克斯，在一次由计算机协会主办的关于编程语言历史的会议上发言，发言的主题是软件设计。[34]他指出，某些编程语言和操作系统拥有“粉丝俱乐部”——在他看来是“狂热分子”，而另外一些则没有。他问道：“哪些产品没有粉丝俱乐部，但做得非常成功、应用广泛、卓有成效、贡献巨大？”他列举了 COBOL、OS/360 以及微软的 DOS 操作系统等软件。而另一方面，他也列举了一些拥有粉丝俱乐部的软件，包括 FORTRAN、Pascal、C、Unix 以及 Macintosh 操作系统等。

布鲁克斯说，区别在于，那些拥有粉丝俱乐部的语言和操作系统“最初都是为了满足一位设计人员或一小部分设计人员的需求而设计的”，而那些没有粉丝俱乐部的成功产品“是为了满足大多数群体的需求而设计的”，它们是“在内部完成的产品过程”。所以，布鲁克斯问道：“有关生产过程，这能告诉我们什么？”接着他回答说：“他们生产的是有用的东西，而不是伟大的东西。”

第 4 章

打破巨型计算机的控制：Unix 和 C

肯·汤普森迫不及待地想要阅读 IBM 360 大型机的程序员手册[1]。自从 1964 年 4 月第一周，IBM 发布该产品，他每天都打电话向 IBM 设在圣何塞的办事处索要手册。得知可以拿到手册的那天，汤普森立即跳上自己那辆 1959 年的深蓝色大众汽车（一款经典的甲壳虫），沿着尼米兹高速公路由伯克利冲向了圣何塞。

当时，汤普森还是加州大学伯克利分校的一名大学生。在驶往圣何塞时，他还不是 IBM 的拥趸。鉴于当时的时间和地点，这一点也不奇怪。几个月后，伯克利市参加自由言论运动的学生走上街头，抗议学校管理部门的“专制”。约翰·肯尼迪遇刺一年之后，全国各地的大学生开始深刻地意识到美国的种族

问题和暴力问题，而伯克利的大学生则引领着这一思潮。苏联“史普尼克号”人造卫星发射之后，美苏开始了太空竞赛。在美国政府和企业的引导之下，大学课程开始向数学和自然科学方面侧重，同时学校也加强了对学生领袖的管教和控制。用抗议者的话来说，“系统”和“机器”是邪恶的代名词。一张颇具代表性的传单上公然宣称：“国家机器不仅无视我们的存在，还威胁和伤害我们。我们要关闭这台机器。”[2]（不过，并非每句话都是这么冰冷激进，传单结尾写着：“到中午的集会来吧，琼·贝兹①也会到场。”）

在 1964 年之前，IBM 被视为企业技术独裁的象征，因此受到许多学生的鄙视。不过，汤普森对这家公司表达不满的方式比较特殊。这名工科学生并没有选择在街头抗议，而是在学校的计算机中心——他长期逗留的地方，也是像他一样有技能的程序员的地盘——用技术来表达自己的抗议。汤普森对 IBM 的鄙视主要是技术文化方面，他认为 IBM 在这方面有所欠缺。多年后，他以嘲讽的口吻对那些被 IBM 八面玲珑、衣冠楚楚的推销人员灌得酒足饭饱的企业经理说，IBM 销售的不过是“漂亮的蓝盒子”。汤普森发现 IBM 的市场营销很“低俗”，而且它的计算机设计经常“完全忽略了与计算机打交道的人”——工程师和程序员。

当时，汤普森不仅熟悉 IBM 大型计算机，对以 Digital Equipment 公司的小型机为代表的新计算设备也不陌生。尽管 Digital Equipment 还未成气候，但其 PDP 系列从做出模型开始就开创了另一种计算风格。这些小型机成本更低，体积更小，放置在用玻璃隔离、装有空调的房间，由受过训练的“操作员”使用，与大型计算机代表的文化完全不同。相比较而言，PDP 小型计算机更小、更开放、更加诱人，权限不受企业审核和级别的限制。小型机最先被应用于科学研究、工程开发和学术研讨，它降低了那些充满好奇的人们用计算机展开实验的成本和门槛。年轻的研究员和学生也能亲手操作计算机。对他们来说，这些小

① 琼·贝兹是 20 世纪 60 年代美国著名的民谣歌手和社会活动家，是美国民歌复兴运动的重要成员之一。——译者注

型计算机可谓应运而生，同时也为黑客提供了接近计算机的途径。当 IBM 表现出会计账簿似的严苛风格时，小型计算机似乎更多地体现了实验室工作台般的非正式、随意的风格。

拿到 360 大型机的手册后，汤普森便把它放在身边，在从圣何塞回来，途经弗里蒙特、海沃、奥克兰最终到伯克利的途中，他一边翻看，一边开车。这可不是个好习惯，但他无法控制自己。越读下去，他就越确信这就是 IBM 傲慢及忽视程序员和工程师的证据，而这一点将让其自食其果。对 IBM 来说，程序员和工程师本应是最重要的人。他回忆起 360 大型机的操作系统时说道："太大，太慢，太复杂，太变态。"数据地址在内存中的处理机制极其复杂。它好像一艘爬满了甲壳动物的超载的船，这是 IBM 销售人员的梦想促使设计人员弄出来的脑残设计，他们费尽全力想要满足所有类型的客户，去迎合尽可能大的市场。

"我差点出车祸，因为太兴奋了！"汤普森在 2000 年秋天回忆道，"我心想：'就是它！'这台机器将会把 IBM 带进马桶！我确信它将成为计算机市场中的 Edsel[①]。"

贝尔实验室位于新泽西州默里希尔。在离其计算机科学实验室较远的一个临时会议室里，汤普森坐在一张旧长椅上，浅啜着咖啡，回忆起往事时忍不住轻声笑了起来。他身材高大结实，穿着牛仔裤、运动鞋和工作服，头发从高高的发际垂到肩膀，蓄着长长的络腮胡，目光炯炯有神。同事对汤普森的评价是，他就像"肥胖版的拉斯普金[②]"。

从某种意义上说，汤普森年轻时对 IBM 360 的评价可谓一针见血。360 给程序员带来的负担令人望而却步，操作系统像怪物一样难以生产和维护。但是，

① Edsel，艾迪塞尔，福特公司 20 世纪 50 年代推出的一款汽车品牌。由于决策失误，市场极其惨淡，给福特带来了极大的负面影响。——译者注

② Rasputin，拉斯普金，俄国尼古拉二世时的神秘主义者、俄国沙皇及皇后的宠臣。——译者注

当时汤普森是以优秀而纯粹的软件大师和开发者的眼光来阅读这份程序员手册的。客观来说，IBM 的设计包含技术方面的权衡折衷，这对于公司计划拓展产品线，进而全面覆盖“360 度计算”是很有必要的，因为它能使客户在 360 大型机系列中平稳升级，同时又不必为财务、人事及制造等部门重新开发软件。

这种设备特征符合当时大型机时代的市场，对此，汤普森的认识有失偏颇，但是他确实找到了大型计算机的短处。1969 年，汤普森与他在实验室的长期搭档丹尼斯・里奇密切合作，开发出了 Unix 操作系统。高效、简洁、精致的 Unix 与 IBM 360 操作系统形成了鲜明的对比。最初，Unix 仅作为工具供贝尔实验室的计算机科学家使用，由程序员编写，为程序员服务，没有商业计划，也没有营销策略。但之后，Unix 逐渐走出了贝尔实验室，进入了大学和工程界，并被广泛接受。工程师、研究人员和计算机学者都爱上了 Unix，因为对于实验来说，它是一个理想的媒介，他们能轻易地按照自己的喜好来组合软件工具，并组成操作系统，用一位贝尔实验室科学家的话来说，就好像把花园浇花用的软管对接就能使其变长一样。

Unix 操作系统引领了抵制大型计算机的革命。它是一种分散式软件，成为分散计算的混合搭配工具。不可否认，分时共享在计算机个人化的方向上迈进了一步，因为中央主机分配资源时会让人产生独占操作的错觉。但最终实现将计算能力置于人们的股掌之间——一人一机的，是发端于 20 世纪 70 年代中期，到 20 世纪 80 年代成为一股主要力量的个人计算机。处于个人计算和大型机之间的是小型机，20 世纪 70 年代初期，随着价格的降低，小型机迅速席卷了学术界和商业领域。突然之间，采购小型计算机不再是一个耗资数百万美元的决策（那需要董事长办公室同意）。在最初的 10 年里，Unix 主宰了小型机。

Unix 创造了一种技术理念。虽然最初是为分时计算机而设计的，但 Unix 并不仅仅是一个程序运行环境，而是像丹尼斯・里奇所描述的“一个可以连接周围的系统”。这种公共计算的观点直接成为 20 世纪 90 年代“开放源码运动”——全世界的程序员分享源码并共同协作——的先驱。开放源码的典型产

物 Linux 就是源自 Unix，而且在因特网上作为服务数据节点的绝大多数服务器都在 Unix 的各种衍生版本上运行，这绝非偶然。

颇具讽刺意味的是，作为精英主义催生的项目，Unix 系统由程序员开发，也服务于程序员，并未将贝尔实验室工作人员之外的普通用户考虑在内，却成为公共计算的试金石。此外，同样的文化、同样的程序员还开发出了最流行的计算机语言之一：C 语言。C 语言是里奇在汤普森的帮助下，特意为 Unix 的实施而开发的。肯·汤普森是创意的支点，大多数好点子都是他想出来的，他是一位无与伦比的程序员。他的同事兼前任老板 M. 道格拉斯·麦克罗伊说，在“那个点石成金的年代”里，汤普森是“首席点金师”[3]。

1943 年，肯·汤普森在新奥尔良出生，但在那里待的时间并不长。他的父亲刘易斯·埃尔伍德·汤普森是一名海军飞行员，所以他们通常每一两年就要搬一次家，住过的地方远远不止华盛顿、俄勒冈、加利福尼亚、得克萨斯、日本、意大利。汤普森从小就表现出了对机械的痴迷，尤其在需要冒险的时候更是如此。得克萨斯州金斯维尔市到处都是油田，工人们用短波无线电联络，汤普森流连在当地的无线电商店内，混迹于那些无线电爱好者中间。当维修人员从店里出发去油田的时候，他也紧跟着，爬上钻塔去回收那些损坏的无线电设备。汤普森和家人在意大利那不勒斯待了 3 年，直到他读完高中二年级的时候才离开那里。“那是我们待得最久的地方。”他回忆道。那时他已经开始对更加精巧的东西感兴趣，例如制作一个能拾起物品的电子机器人，或者将管子和咖啡罐焊接起来做成火箭，向那不勒斯的夜空发射。事实上，他毕生都热衷于飞行，正如他的父亲一样。48 岁时，他在俄罗斯花费 12 000 美元，只为驾驶米格-29 喷气式战斗机做出各种动作，如翻滚、画圆、殷麦曼翻转、锤头失速以及翻转失速等，他认为这项体验是重力和肾上腺素各占一半的冒险。尽管汤普森并不以公众演说家或作家的身份闻名于世，但他却忍不住把这次体验写成日

志，并发布在他的网站上："我们起飞了，喷射引擎全速运转，就像有人在踢我的肾脏。"[4]

汤普森上高三时，他的父亲被调回到美国在圣地亚哥的海军基地。汤普森的父亲在俄克拉荷马州的金菲舍尔区长大，在俄克拉荷马大学读了一年。不过，当时正是大萧条时期，他由于缺钱而辍学，随后加入了海军。为人父后，他打定主意，要让三个孩子中最聪明和最年幼的那个接受大学教育。"我是否应该去上大学，这从来都不是问题，问题是要上哪一个。"肯·汤普森说道。到底是加州大学伯克利分校还是麻省理工学院？学费和路费的差异最终促使他选择了公立大学。

在伯克利上学时，汤普森这个电子爱好者自然而然地选择了电子工程专业，被伯克利计算机中心吸引。他很快开始在计算机中心长时间地工作，当然，在某种程度上这也是为了生存，能帮助他支付学费和食宿。汤普森认为自己更像一个为谋生而工作的职业计算机工作人员，而非一名学生。他觉得自己获得电子工程学位"差不多算是个意外，其实什么也没干"。汤普森说，他的导师签约雇用了他，并给予他在计算机中心工作的学分。1983 年，他和里奇获得了图灵奖——计算机科学领域的诺贝尔奖。汤普森解释道："我是一名程序员，在我的 1040 表①上，我就这样填写自己的职业。"[5]他对各种编程都很着迷，正是在伯克利计算机中心，他被编程迷住了。多年后，坐在贝尔实验室的办公室里，他这样描述那种吸引力：就好像工匠打造物件不必为成本和采购原材料而烦恼。"就像盖房子却不必购买水泥一样，"汤普森说道，"你创造自己的世界，你拥有这一切，从来都不需要离开这个房间。"

每当夜幕降临，汤普森就会在伯克利计算机中心随心所欲地折腾，他可以编写程序、安装机器，做他想做的任何事情，一般是玩游戏。他有一个爱好，那就是在 4×4×4 的棋局里玩一字棋。跟传统的游戏相比，它要求玩家（人或者

① 1040 表，美国的纳税申报单。——译者注

计算机程序）计算多得多的招数。就好比将 4 个 16 方格的一字棋棋盘叠在一起。游戏者要在任何一个单独的棋盘上排成连续一行的 4 个方格或者用 4 个棋盘中每个棋盘上的方块来排列。由于要判断树和分支结构，复杂的一字棋游戏需要求助于数学博弈论专家来破解迷局。

那时，汤普森认识了伯克利分校数学系一流的博弈论专家埃尔文·波利坎普。这位数学教授兼计算机中心黑客接受了汤普森的挑战，在接下来的几个月里，在黎明前的几个小时里和他玩多维一字棋。汤普森绞尽脑汁，在计算机上编写好程序，然后在凌晨 2 点左右把波利坎普从床上叫起来。“他过来和我玩游戏，由我和我的电脑来对抗他和他的大脑。”汤普森回忆道。白天，波利坎普是个精力充沛的人，一到晚上，他就常常睡眼朦胧，摇摇晃晃。“但我们从未打败过他，”汤普森说，“我被气疯了，舔舐着伤口重新编写程序，但是他再次出现，然后再次大获全胜。”

不管怎样，对汤普森来说，正是那些在夜间玩的一字棋游戏把他带进了贝尔实验室。在他毫不知情的情况下，波利坎普推荐了他，于是他成了贝尔实验室招聘人员的目标。当贝尔实验室的招聘人员出现在汤普森在伯克利的公寓时，汤普森说他十分乐意去东部免费旅行，正好想去拜访几个朋友，但是他对受雇于电话行业垄断者的研究实验室并不太感兴趣。到达贝尔实验室后，他接受了来自多个部门的代表为期两天的面试。汤普森说，大部分研究部门对计算机都持相同的态度，就像 20 世纪 90 年代大多数美国公司对待互联网的态度那样，他们知道自己不得不接受新技术，却基本上什么都不懂。但是，贝尔实验室的计算科学中心，是个绝对的例外。它似乎更加开放、不拘一格，这里汇集的研究人员都渴望探索科学计算这一新兴的领域，无论在哪里都有可能成为领军人物。“那是个不同寻常的地方，”汤普森回忆道，“我认为他们很棒。”

丹尼斯·里奇从事计算工作及进入贝尔实验室的经历则完全不同[6]。里奇

在舒适的纽约郊区长大，他的父亲阿里斯泰尔·里奇是贝尔实验室的一位科学家。还在哈佛大学读大一时，里奇就参加了珍·萨梅特介绍 UNIVAC 的讲座。他的兴趣由此被激发，后来选修了工程科学 110，这是介绍计算机的入门课程。他还参观了当地的 IBM 办事处，并在那里获赠了一份 FORTRAN 指南。他早期的编程大多是应用数学程序，例如，采用微分方程计算液体通过圆筒的流量。虽然里奇是经由数学而非电子工程开始从事计算工作的，但是他发现比起理论化的数学模型，他对实实在在的计算机更感兴趣。他继续攻读数学博士学位，然而，尽管完成了论文并通过了答辩，却没有获得博士学位。之后，他再也没有兴趣为获得学位而去做那些文字工作了。30 多年之后，忆起这段往事，他说这种事“年轻的时候义无反顾，现在想来未免遗憾”。无论如何，当时他对计算机的兴趣太过强烈，也就无暇他顾了。

在哈佛上学时，里奇在麻省理工学院还有份兼职，负责做分时共享实验。身材修长、蓄着胡须的里奇，有好几次在贝尔实验室接受采访时，都深情地回忆起那段日子。他强调，早期的兼容分时共享系统（CTSS）尽管在很多方面都很原始，但用户已经可以与在同一终端上的其他用户建立开放的连接，这相当于今天互联网和在线服务领域盛行的“即时通讯”。后来，里奇加入了麻省理工学院的 MAC 项目——由国防部先进研究计划署（ARPA）提供资金的一项大型分时共享系统。CTSS 和 MAC 项目是开发大众计算技术和研究人机交互的实验，计算机资源按当时的标准以极小的循环周期被共享。例如，第一个 MAC 项目系统可以在同一时间处理 30 位坐在输入终端前的用户的操作。MAC 项目，特别是它的下一代分时共享系统 Multics①，对那些从事这项研究的人员和计算机技术本身都产生了深远的影响。

离开哈佛后，里奇受雇于政府的圣地亚国家实验室，负责武器研究和测试。

① Multics，Multiplexed Information and Computing System，多任务信息与计算服务，是 1964 年由贝尔实验室、麻省理工学院及美国通用电气公司共同研发的一套安装在大型机上的多人多任务操作系统。——译者注

“但是，那时已经快到 1968 年了，”里奇说，“设法替政府制造原子弹似乎已经不符合时代潮流。”于是在 1967 年，里奇转而加入贝尔实验室，比汤普森晚了一年。很快，他们俩都陷入了 Multics 计划的困境[7]。CTSS 和最初的 MAC 项目分时共享系统都在 IMB 大型机上运行，而且麻省理工学院和贝尔实验室均与 IBM 有着密切的联系。麻省理工学院和贝尔实验室也都致力于分时共享系统。然而，当时 IBM 似乎并未意识到这是计算机领域的一个重要的新方向，只将其视为学术上的好奇而已。IBM 大型机是为长时间、不间断地运行批处理任务而设计的，其目标并非分时共享系统那种动态的交互模式。1964 年，IBM 被告知，它的新型 360 主机不适合分时共享系统，因此 MAC 项目选择了通用电气的机器。后来，贝尔实验室很快也采用了通用电气的计算机，以满足其分时共享系统的需要。

1965 年之前，在 Multics 项目上，贝尔实验室一直与通用电气和麻省理工学院合作。当时，Multics 是最具野心的分时共享系统的尝试，设计支持 1000 个连接的终端和 300 名用户同时作业。通用电气提供硬件设施，贝尔实验室和麻省理工学院致力于开发软件。美国电话电报公司未能涉足计算机业务，因为 1956 年它曾与当局达成一项反托拉斯和解，并在其中承诺不涉足“公共载波和通信服务”之外的行业。但是贝尔大妈①明白计算机技术对于它的未来至关重要。对于电话公司来说，即便是电子交换系统，其本质上也是一台大型计算机。现在的数字交换机使用了一些最复杂、精密的软件，就像交通警察那样指挥着无穷的语音和数据信息流。美国电话电报公司始终未涉足此行业，直到 1984 年它同意被拆分为多个独立公司之后，才得以按照自己的需求自由发展计算机技术，并从事类似于 Mtulics 的项目研究。而分时共享看起来就像是贝尔系统的另一个网络。

Multics 是对大规模高效率分时共享的一次努力。这是一个雄心勃勃、极

① Ma Bell，贝尔大妈，美国电话电报公司的绰号。——译者注

具价值的目标，但是汤普森说 Multics 患上了糟糕的“第二系统综合征”，分时共享的理念中一次性添加了太多的新东西。试图维持三个地理上分散、文化背景毫不相同的机构进行合作必然毫无益处。1969 年，贝尔实验室断定 Multics 是一个代价高昂的错误，于是退出了该项目。然而，贝尔实验室团队成员都很喜欢麻省理工学院风格的分时共享技术。里奇解释道，在 CTSS 和 Multics 等系统中，用户可以共享从计算机到其程序与信息存储的同一个连接。这种对一个共享资源的同时使用，意味着“合作和沟通将变得更加简单，你工作的内容对所有人都是透明的”。

事实上，创造 Unix 是为了从 Multics 手中争夺协同计算环境，虽然它简单又廉价。和 Multics 相似，Unix 是为基于分布式系统概念的交互系统而设计的。Unix 的树状结构，命令名称和命令解释程序，或者外壳①，都与 Multics 非常类似。“人们普遍认为，Unix 的产生是为了应对 Multics，其实，两者之间有着深厚的文化渊源。”里奇这样说道。

甚至这个操作系统的名字 Unics（最初没有“x”，不过很快就加上了，这个名字最初是写在纸带上当标签用的），用创造这个词的贝尔实验室研究员布莱恩·科尼汉的话说[8]，是针对 Multics“非常隐晦的双关语”。他说：“意思是 Mutlics 是很多东西，Unix 就是其中之一。”科尼汉是 Unix 小组的早期成员之一，跟里奇一样曾经参与 MAC 项目。“如果没有 MAC 项目的血统回归，没有人痴迷于此类协同计算，Unix 也就不会诞生。”

Unix 的故事再一次表现了软件和硬件之间的阴阳关系，彼此不同却又相互依赖。一旦压力点出现，即硬件的发展提供了新的可能性，或者需要跨跃式的创新以克服硬件的障碍，软件就会取得突破性发展。Unix 就是通过软件的创新发展克服硬件障碍的例子。

放弃 Multics 项目之后，汤普森和里奇游说他们的上司，希望能得到允许，

① shell，外壳。操作系统与外部的主要接口。——译者注

创建自己的分时系统，并为此购买一台新型 Digital PDP-10 计算机。贝尔实验室管理层拒绝了这个提议，但是这些 Multics 遗少们并没有放弃。多亏了汤普森对游戏的嗜好，令他们手边有可供选择的替代品。利用在 Multics 项目工作时的富余时间，汤普森开发了一款名叫“太空旅行”的电脑游戏。该游戏模仿了太阳系中行星和月球的运动。通过输入指令，玩家能够游弋太空，甚至能在行星上着陆。这款游戏在计算机科学研究人员中极受欢迎，但是，贝尔实验室管理层并不想让 GE 635 大型计算机昂贵的机时被浪费在一款太空游戏上。于是，汤普森找到了一台 Digital PDP-7，虽然它是一台过时的小型计算机，却拥有出色的显示终端，非常适合在屏幕上进行星际旅行。他和里奇为 PDP-7 重写了太空旅行的代码，对于性能较差的机器而言，这是个不错的方法。

有一段时间，汤普森、里奇和贝尔实验室的其他程序员一直思考如何为分时共享设计合理的文件系统。里奇称之为“粉笔文件系统”，因为它起源于实验室里一系列马拉松式的黑板讨论。1969 年夏天，汤普森开始在 PDP-7 上编写 Unix 的第一个版本，共有 4 个主要部分：操作系统核心、外壳、编辑器以及将程序翻译为机器语言的汇编程序。那时，刚好他的妻子邦妮带着他们的儿子去加州探望汤普森的家人，要过一个月才回来。在妻子离开的那段日子，汤普森给自己规定每周解决一个部分，这就是 Unix 的由来。汤普森认为，其主要意义在于实现了自己长期构想的分时操作系统的瘦身版。事实上，作为用于双站分时共享系统的混合编程，Unix 的出身可谓卑微。“它就诞生在这样一个破得让人不敢相信的硬件上面。”里奇说。

硬件的掣肘确保了 Unix 的精致、紧凑和简单。对于汤普森、里奇和 Unix 来说，在提出请求后没有被允许购买更大的机型 PDP-10，也许是因祸得福。“我认为它确实是样好东西。”科尼汉说道。他于 1969 年加入贝尔实验室，并成为 Unix 的早期成员之一。

1973 年 10 月，在一次会议上首次对外展示 Unix 时，里奇和汤普森这样描述道："也许 Unix 最重要的成就在于，它表明一个强大的交互应用操作系统既不需要在设备上也不需要在人力上花费高昂的代价，它可以在价值仅为 4 万美元的廉价硬件上运行，而且主要的系统软件只花费了两个人不到一年的时间。"[9]

开发 Unix 的过程也能说明 Unix 的精益效率，一个合作多年的小团队按照自己的节奏工作，不受市场调查或客户需求的束缚。"我们没有考虑太多，"汤普森说，"做这一切都是为了我们自己，从某种意义上来说，我们就像自大的独裁者。"

"就其本身而言，确切地说，Unix 并非一个目标。"他接着说，"这不像曼哈顿计划，要去轰炸某个地方，那就是你的目标。如果一开始 Unix 真有目标的话，那么它就是一件工具，开发它是为我们的研究提供支持，为了我们想要做的其他事。"

所以，Unix 是技术酝酿渐进过程中的产物。这个过程允许 Unix 按照一个远离商业压力的时间表发展。没有强加的功能，而且由于没有客户意见的烦扰，大修大改也是可以的。他们可以将研究得来的好想法原样投入到操作系统中。

1972 年实施了软件"管道"这个概念，这对 Unix 来说是个重大的突破。道格·麦克罗伊是贝尔实验室的科学家兼管理者[10]，长期致力于推动连接程序的设想，就像在数据流中那样。多年以来，贝尔实验室的研究人员一直在努力开发出有助于克服传统批处理过程限制的软件。例如，20 世纪 60 年代初期，贝尔实验室的程序员开发出了一种用于数据串操作的语言 Snobol，它在列表处理方面与约翰·麦卡锡开发的 Lisp 语言算是远亲。1964 年 10 月贝尔实验室打算加入 Multics 项目之时，麦克罗伊曾在内部备忘录上写道："我们应该找到连接程序的办法，就像连接浇花水管那样，当有必要换一种方式报告数据的时候，就接入另外一段管道。"[11]

1972 年，麦克罗伊敦促汤普森把流处理纳入 Unix。汤普森回忆，麦克罗

伊画了一些草图来表达他的想法。这些图上有复杂的网格，通过网格，每个程序的输出都可以提供给其他任意程序，没有次数限制。每节水管可以有很多连接器，就像是炼油厂采油树阀门的微型软件版。这种把简单事情复杂化的做法让汤普森颇受打击，却也给他留下了深刻的印象。“我被麦克罗伊设计中的这种预想惊呆了。”他说。因为这与他自己的软件方法有着天壤之别。“我使用连续近似法，那是我的编程方法。我只是把一个程序推向我想要实现的结果。”

在汤普森看来，他上司的这个设计只是一个理论幻想，完全低估了软件不可预测的复杂程度。“我这样回答他：‘有一大堆程序！你甚至不知道哪些在运行，更不必说它们要如何运行……’我觉得他的想法不切实际。我在心里已经认定他错了。”

尽管麦克罗伊的方法看起来有些误入歧途，但他关于数据流的概念却让汤普森颇感兴趣。老板不断在催促，而他一直在思考。汤普森回忆道：“一夜之间，也不知道怎么回事，我想到了让它变成一个维度的办法。”也就是说，汤普森决定不再让许多其他程序的输出任意输入到某个程序，而是令其一一对应。一个程序的输出成为另一个程序的输入，这就像是“花园浇花用的水管”一个个连接起来构成线性的组合，而不是堆在一起。他将麦克罗伊设计中复杂的一般情况削减成简化的特殊情况。为了实现一个美好的设想，在编程术语中将不会有回旋的余地。“如果没有其他选择，就只有成功了。”汤普森说。回首往事，如今已是普林斯顿大学教授的科尼汉还忍不住惊叹：“这简直是爆炸式的创新。它开启了各式各样的大门。”

在 Unix 中实现管道的工作花费了三个晚上才完成——第一晚是汤普森一个人，之后的两个晚上里奇也加入了进来。最初，汤普森赋予操作系统的外壳一个工具，用于创建允许一个程序与另一个程序直接通信的通道，这样就移除了软件在操作系统中运行所需的中间步骤。以往，程序是独立存在的实体，可

以运行然后转到某些暂存文件中，等候新的指令，这些暂存文件就像是软件围栏。在流处理机制中，随着程序不断地接收来自其他程序的数据流，一切变得流畅而灵活。后两个晚上，为了更好地应用流处理的概念，汤普森和里奇通过修改内部规则、表示法和工具等，对 Unix 进行了一次大修。

新泽西郊区贝尔实验室六楼的一个房间里，在低矮的天花板下，他们借着荧光灯，佝偻着身体，在 PDP-11 计算机的键盘前工作，房间号是 2C644，就像老式计算机的内存地址。有时，他们也在家里工作，两人都有远程终端。他们通常很晚才开始工作，汤普森总是在贝尔实验室的午餐厅下午 1:15 关闭前一两分钟才到，然后一直工作到凌晨三四点钟。汤普森解决了流处理的难题后，他们的工作进程便加快了。为了充分利用流处理，他对 Unix 的特点不断进行调整。这是一个激动人心的冲刺过程。“或许那是我一生中最棒的三个晚上。”汤普森说。

这份工作带有开发者汤普森的个人印记，他以快速、简洁和富有想象力而闻名。麦克罗伊说：“汤普森的代码读起来令人愉悦，十分简洁，很明显是经过深思熟虑，而不是靠改正缺陷得到的。”当改善问题的好办法出现时，汤普森就会摒弃旧的代码从头开始编写，而不是试图去做些修修补补的工作。他尽量避免使用宏命令、图形辅助代码书写等编程捷径。1975 年，英国程序员乔治·康乐瑞斯开发出一种可视化产品，允许程序员在图形显示器屏幕上查看代码，替代汤普森为 Unix 开发的代码编辑工具。在贝尔实验室，汤普森使用电传打字终端进行编程，只有键盘，没有屏幕，依靠打印出来的资料进行工作，并且要记住所有的改动。汤普森去伦敦的时候，康乐瑞斯向他展示了自己的屏幕编辑器，他记得汤普森这样回答：“嗯，我见过像这样的编辑器，但我并不需要。编辑的时候，我不想去看文件的状态。”[12]后来，康乐瑞斯决定将他的可视化编辑器命名为 em，是“短命编辑器”（editor for mortal）的缩写。

汤普森的程序很少有注解；他编写的代码不说自明，简洁明了。里奇谈到他时说，汤普森能够对手头的任务“提出令人惊奇的、极具创造力的正确算法”。

汤普森拥有一项早期的软件专利[13]，于 1967 年提交，1971 年发布，是一种新型的文本匹配算法。在该专利 15 页的陈述和编程说明中，描述了一种通过计算机快速识别文本模式的方法。通过采用这种算法，程序可以同时处理几项并行的检索功能，比同一时间只能处理一个的串行方法快多了。汤普森用于快速识别和编译数据的编程技巧，有点像编译技术的先兆，这一飞速发展的技术使互联网程序运行得更快。

汤普森还将其快速遍历逻辑树的算法创新应用到了数学游戏中，例如 moo 和 chomp，以及后来的国际象棋。1980 年，汤普森为一台弈棋计算机编写软件。他和贝尔实验室的同事乔・康顿花了几个月时间装备这台名叫贝拉的计算机，而要做到这一切，需要汤普森巧妙地运用一大堆专用计算机语言，还需要动手接线。“汤普森钻到硬件里面了，”里奇说，“他能实打实地做东西。”贝拉是第一台达到国际象棋大师水平的计算机，为以后的机器（如 IBM 的“深蓝”等）设定了模式，“深蓝”在 1997 年击败了国际象棋世界冠军加里・卡斯帕罗夫。

他的老板麦克罗伊指出，1972 年，在汤普森开始将流处理纳入到 Unix 中的第一个晚上就已经戏剧性地证明了汤普森“洞悉项目全局的神奇能力”。麦克罗伊回忆道，他不仅将流处理付诸实施，而且改进了操作系统外壳去充分利用管道，随后还修改了 Unix 的某些公用程序。那些零星想出的东西，麦克罗伊解释说，要是其他程序员，得花费多得多的时间才能实现。他说：“一个谨慎的项目经理，会给那些工作分配几周的时间。”

最重要的是，管道让 Unix 不只是一个操作系统。Unix 环境的出现，代表了一种全新的编程思路，即“工具哲学”。利用管道机制，Unix 鼓励程序员把程序视作灵活的工具，能够与其他程序组合共同工作或者构建软件应用。这就是在软件中引入的 E.F.舒马赫“小即是美”的思想，与那些把相关功能全部封装在一个程序里面的观点形成了鲜明的对比，后者诸如360型主机的操作系统，

或者后来的微软 Windows 操作系统，还有微软的 Word 文字处理软件和 Excel 电子表格等大规模应用程序。

拥护工具方法的程序员也经历了观念的转变。布莱恩·科尼汉是最早转变观念的一个人，他这样描述自己的领悟："嗨，我并不是在写程序，而是在制作别人能使用和依赖的工具。"汤普森和里奇被授予图灵奖时，颁奖词是这样写的："天才的 Unix 系统能够使程序员借助他人的成果"。

在普林斯顿大学的办公室里,科尼汉用一个简单的例子演示了如何在 Unix 中利用管道连接不同的软件工具。一开始，科尼汉就想查出有多少人接入学校的网络，并且正在和他使用同一台网络计算机。他敲入 Unix 命令 who，然后屏幕上出现长长的列表，显示出此时连接到中心计算机的工作站，每一行包含着分配使用这台计算机的用户名字。他说，以前在贝尔实验室时，who 这个工具是快速找出周围有哪些人的方法。但是普林斯顿的用户列表包含了数百个名字。为了搞清楚网络中有多少人，科尼汉用管道将 who 的运行结果输入到了一个计数程序，这个程序实际上计算 3 个内容：行数、单词数和字符数。该程序指令为：who | wc（单竖线是管道的标志）。计数程序的计算结果是 428 行，因此，在普林斯顿的同一台中心计算机上共有 428 个连接。然后，科尼汉说，要是他想找出这些连接中是否包含某个朋友，例如乔·史密斯的一个或者多个连接（一个用户可以拥有数条连接），没问题，用 Unix 文本模式搜索工具 grep 就可以了。要查明乔当前是否登录，可以用管道：who | grep Joe Smith。如果要查明乔·史密斯那天在中心计算机的登录次数，就引出了另一个管道：who | grep Joe Smith | wc。

管道能够提高灵活性和编程速度。1973 年，在一次经典的工具方法展示中，贝尔实验室的研究员史蒂夫·约翰逊拼凑出了一款拼写检查工具，其中用到了 6 个 Unix 工具，包含一些排序和转换工具，以及一个类似字典的单词列表。总共花了他大概 1 分钟。"除了管道的单竖线之外，史蒂夫根本就用不着写一行代码。"科尼汉回忆道。Unix 环境赋予程序员更大的自由去快速地添加、去除

或者组合工具。实验软件的不同组合变得更加容易。举例来说，在 Unix 的世界里，字处理程序中的拼写检查和字数统计功能均为单独的程序，因此可以有所选择。不过，在微软的 Word 中，也有一个拼写检查工具和一个字数统计工具，但这是微软选定的那个。

有时候，这两种截然不同的方法就好似对立的政治阵营或宗教派别的斗争。对于某一类程序员而言，Unix 的工具哲学凭借其承诺自由实践，可以不受限制地选择，而被视为是实用的和高尚的。这一点正是这类程序员的理想，他们崇尚自由、松散的环境，而且就像汤普森形容自己的风格那样，通过不断接近的方式工作，将程序一点一点地推进直到最终完成设计。汤普森信仰这种文化一点儿也不意外，而这种文化也正是在他的帮助之下创造的。关于微软的做法，他说："我觉得那种理念不可取，对用户不太尊重。它把用户当成笨蛋了，所以做出来的所有东西都不可编程。"

然而，对另外一些人来说，Unix 的工具系统只是产生混乱和多余工作的公式。这些程序员和用户更喜欢享受微软提供的基本软件，能够在此之上建立自己的应用，或者只是干自己的工作。他们并不想依靠自己来发掘软件。就连里奇也承认 Unix 只是一种工具方法，不适合所有人。"工具方法背后的理念是每个部件都要足够简单，这样人们才能了解它们，"他解释道，"但是我们可能没有成功地把事情做到我们希望的那样简单。并不是每个人都有能力做这类事情。这种模式或许还很遥远。"

如果回到 20 世纪 70 年代初，任何人都无法想象 Unix 竟然能发展到如此地步，当时这个新生的操作系统在贝尔实验室之外几乎不为人所知。它的第一次走向实用是为贝尔实验室起草专利申请书而开发的一款文字处理系统，目的在于减轻 3 个专职打字员的工作量。随后，Unix 开始在一些大学的研究人员中传播，当他们听说这个精简、高效的操作系统之后，觉得它似乎是小型计算机上协同工作的理想方案。由于反垄断法的规定，贝尔大妈被强制为其技术设定许可，这项政策确保了一些创新技术的广泛传播，不单是 Unix，还有晶体管、

激光等。里奇、汤普森以及其他一些人呼吁以极低的价格与大学签订 Unix 的许可条款，并向大学公开；美国电话电报公司的律师们看到如此宽松的条件对自己没什么坏处，而且除了在学校落一个好名声之外，从这个软件研究项目中也无利可图，于是表示同意。

大学的研究人员喜欢上了 Unix，在美国一些领先的大学计算机科学系，诸如加州大学伯克利分校、斯坦福大学、卡内基梅隆大学等，Unix 逐渐普及起来，而且在海外，主要是英国，也站稳了脚跟。Unix 通常是在 PDP-11 小型计算机上运行，从根本上降低了计算成本，不仅仅是金钱，还有程序员的时间。据伦敦大学玛丽皇后与韦斯特菲尔德学院的荣誉教授乔治・康乐瑞斯估计，在 Unix 上面设计和调试程序，或许比在批处理大型计算机上快 10 倍。1973 年年末，这个学院使用了欧洲首个 Unix 系统。

贝尔实验室的许可通过公布操作系统的源代码，公开了 Unix 的知识内容。“源代码”都是人们能读懂的程序指令，不同于机器可以理解的由 1 和 0 组成的程序“二进制代码”。Unix 的使用手册中有很多清晰、准确的代码范例。它们是为专家编写的，即便是熟练的程序员通常也需要多读几遍才能理解，但是，它们的覆盖面很广，全面记录了该操作系统的每一个方面。Unix 手册中甚至还包含了有关软件“漏洞”的章节，这种坦率和真诚让人无法想象它是出自商业计算机公司之手。20 世纪 80 年代，一切都变了。美国电话电报公司试图将 Unix 变成常规的软件产品，竞争者也纷纷跳出来争相推出各自的 Unix 版本，市场和技术争夺开始白热化，这被称为“Unix 战争”。但是，在整个 20 世纪 70 年代，Unix 是与众不同的文化。对于那些大学里年轻的计算机科学家而言，传统计算的制约令人不满，Unix 的环境则令他们兴奋，甚至是上瘾。“就像在操场上发放海洛因一样。”科尼汉说道。

C 语言是为 Unix 开发的。它们起源于同一个地方，分享着相似的智慧。

事实上，它们是由相同的合作伙伴创造的。“Unix 是肯・汤普森在丹尼斯・里奇的帮助下开发的，而 C 语言是丹尼斯在肯的帮助下开发的。”科尼汉说。科尼汉为 Unix 作出了重要贡献，还与里奇合作编写了《C 程序设计语言》。

C 语言的起源可以追溯到麻省理工学院和 MAC 项目[14]。它的“祖父”是由英国科学家马丁・理查兹设计的 BCPL，20 世纪 60 年代他曾经在麻省理工学院短暂任职。BCPL 是一个小型语言，作为 Multics 项目的衍生物开发出来；Multics 的主要语言是 PL/I，它尝试将 FORTRAN、COBOL 和 Algol 的元素结合在一起，但过于松散。汤普森在完成 Unix 的首个版本之后，用汇编语言把它写出来，使之适应自己正在使用的小型计算机。之所以采用汇编语言编写，是因为最初用来运行 Unix 的机器性能太差，内存严重不足。不过后来，汤普森也觉得 Unix 应该有自己的编程语言，这样就可以更容易地编写程序，前提是，这种语言必须能够在性能较差的硬件上运行。他的解决方法是设计一个简化版的 BCPL，称为 B 语言。后来，里奇将其描述为 BCPL 缩水版，运行它的计算机内存少得可怜，就像是“汤普森脑袋里漏出来的”。

BCPL 和 B 都被认为是“无类型”语言，意思是这种语言无法区分不同类型的数据。在老式计算机中，数据片段或者数据块通常都称为“字”，类型都相同。举例而言，较小的机器仅能处理 16 位字，较大的机器能够处理 32 位字。但是，1970 年引进了 Digital PDP-11 小型计算机，这种机器的设计可以识别和处理一种甚至多种类型的数据对象，从而为编程语言定义数据的不同类型开辟了道路，例如，整型、浮点型、字符型、日期型等。1972 年，里奇开发了 C 语言，使程序员看得见并且用得上这种机器的这一能力。

和 Unix 一样，C 语言源自相同的理念，它们都是一种简约的工具，依靠熟练的专业知识设计而成。C 语言是一种相当“接近机器”的语言，它的抽象概念并没有太脱离计算机的理解和操作。C 语言仍然是一门高级语言，因为它并未与特定的计算机结合。然而，C 语言的设计只允许程序员在基本的软件底层（如操作系统、编译器等方面）进行牢固的“系统”编程。在 C 语言之前，

系统编程从未真正在高级语言中完成。C语言之于系统编程，正如FORTRAN之于科学和工程计算，使其与机器的沟通更加容易，在这个原则下，降低了计算过程的壁垒。“正确的工具，正确的时机，确立了自己正确的生态地位。”里奇解释说。

通过让C语言在不同的计算机上运行,里奇和其他一些人努力扩大C语言的影响力。通过这种方式，C语言也将 Unix 的应用从小型计算机扩展到工作站，再扩展到超级计算机。

C语言很少限制熟练程序员的自由，也很少保护缺乏经验的程序员使其避免陷入麻烦。C 语言允许程序员使用“指针”，按照程序员选择的任何方式，无论恰当与否,直接操作内存中的数据,巧妙发挥软件的魔力或者让机器崩溃。“指针有其危险的一面，它们很容易被滥用。”里奇高兴地指出，他暗示C语言就像Unix一样并不适合所有人。10年中，C语言与Unix密不可分，借助Unix操作系统的广泛应用而传播，不过，C 语言也使 Unix 更容易接受、更容易编程，从而有助于Unix的传播。20世纪80年代，C语言闯入个人计算机编程领域，同样成为PC行业的主流语言。“有时候，不经意间，为自己设计的工具也让其他很多人受益。”里奇说。

很明显，Unix 和 C 语言满足了计算机技术发展中某个特定时期的需要，用里奇的话来说就是获得了正确的“生态地位”。时机、运气和偶然因素都发挥了作用，但同时它们也是创新、品味和实用的巧妙结合，因此才能扎根、兴旺并且长久。它们还是不同背景、性格和才能的两个人珍贵友情的结晶。肯·汤普森是个更加纯粹的程序员。“我们前面还有很长的路呢。”汤普森回忆道。丹尼斯·里奇同样也是一位杰出的计算机科学家，不过是另外一种类型的，更稳重，更有条理，更具系统性。在他们两个人中，里奇是执笔人和发言人。像C这样的语言能成为国际认可的技术标准，需要他耐心地把握方向。多年来一直指导这对搭档的道格·麦克罗伊指出，里奇的天分很大程度上是“固有的”、“看得明白的”，而汤普森的才能完全不同，是“真正的天才型”，闪烁着“令

人意想不到的、丰富的创造力”的光芒。

无论这种合作关系究竟是何种关系，两人的才智在工作中常常合二为一。1983 年，在接受图灵奖颁奖的时候，汤普森称与里奇合作是一件“美妙的事”，然后还举出了一些小例子来证明在编码工作中他们的想法有多么相似。汤普森回忆说，在一起工作的10年里，他能记起的两人之间失败的沟通仅有一次。“那次，”汤普森说，“我发现我们俩写下了同样的 20 行汇编语言程序。比较了代码之后，我惊讶地发现竟然一个字都不差！”

汤普森总结说：“我们一起工作的成果，远远超出了我们为各自工作所作的贡献。”与之相符的是，Unix 的工具理念，一个能够让程序员站在他人肩膀上的系统，似乎准确地体现了其缔造者的工作之道。

第5章

为大众编程：从达特茅斯的 BASIC 到 Visual Basic

1957 年，托马斯 · 库尔兹是新罕布什尔州达特茅斯大学一位年轻的教授，同时也是麻省理工学院的常客，因为在那里他可以使用那台大型 IBM 计算机。库尔兹使用计算机的经历要追溯至 6 年前，当时他只是个研究生，在洛杉矶一家政府研究中心打暑期工。库尔兹回忆说，虽然后来他在普林斯顿大学获得了统计学博士学位，毕业后即在达特茅斯大学教授统计学专业，但是这段暑期工的经历还是不可避免地让他进入了计算机领域。当时，计算机还是一个新兴的未知领域，在经济、社会、文化方面的潜力才刚刚崭露头角。但是，库尔兹研究计算机的动机却十分简单。40 多年后，在向那些对编程心存畏惧的人解释编程这一严谨而又耗费脑力的人机对话的乐趣时，他说："码农之乐，尽在一个

‘码’字。”[1]

库尔兹早期编写程序用的是汇编语言，在不同计算机上使用的编程语言也不相同。麻省理工学院计算中心使用的计算机是 IBM 704，因此，库尔兹掌握了这台计算机的分享式汇编语言 SAP。1957 年，FORTRAN 问世，但起初人们对所谓的高级语言存有偏见。很多程序员都认为，FORTRAN 是为那些技术水平不高的编程练习者设计的，真正的程序员都用汇编语言来编程，而且他们认为这样做也能节省宝贵的上机时间。因此，当需要编写一个涉及大量统计计算的程序时，库尔兹选择了使用 SAP 汇编语言。但是，经过几个月的尝试，他认输了。他浪费了“一小时宝贵的 704 机时和自己不那么值钱的大量时间”[2]。放弃汇编语言之后，库尔兹尝试了人们一度不屑使用、效率不高的 FORTRAN 语言。他回忆说：“结果，大概只用了 5 分钟的机时。使用高级语言编程能够节省计算机时间，也能节省自己的时间，这次经历对我的触动很大。”

随后，库尔兹和他在达特茅斯大学的同事兼导师约翰·凯默尼开发出了一种简单易学的高级编程语言，BASIC（Beginner’s All-purpose Symbolic Instruction Code，初学者通用符号指令代码）。当初设计这种语言的初衷主要是为了让文科大学生更容易地使用计算机。1964 年，BASIC 语言首先出现在达特茅斯大学的分时系统中。到了 20 世纪 60 年代末 70 年代初，它已经成为一种在分时计算机系统中普遍应用的编程语言，使用群体广泛，遍及大学生、中学生甚至小学生。对于那些开创了个人计算机产业、自命不凡的年轻新贵来说，BASIC 是他们接触的第一种编程语言，并在很大程度上塑造了他们在这个行业的发展模式。不断涌现的个人计算机产业新秀们，兴致勃勃地对 BASIC 进行增、删、改。他们步伐太快，以致于没有任何标准化组织能为 BASIC 设立官方标准。在新兴的个人计算机产业建立之前那段完全开放的、多姿多彩的创业时期，作为一种编程语言，BASIC 简直太理想了。它简单易学，灵活多变，仅有的一些非正式规则就算打破了也不至于有灭顶之灾。“BASIC 就像是个开放的城市，如同一百年前的上海，”[3]早期的 PC 程序员阿兰·库珀指出，“那里没有法律。”

到了 20 世纪 70 年代末，市面上已经出现了几十种不同版本的 BASIC。微软在 1975 年推出的 Micorsoft BASIC 凭借其巧妙的编程和精明的市场策略在市场上独领风骚，成为这个软件巨头的首个产品。计算机行业的人士都知道，开发编程语言本身并不能赚大钱，因为它说到底只是一种工具。但是，多年以来，微软的策略一直都是通过鼓励软件开发人员开发出能在微软平台上运行的程序，从而建立一种行业标准技术，或者说一种技术“平台”。这种模式始于 Microsoft BASIC，并延伸至 Windows 操作系统。Windows 的成功在很大程度上仰仗了微软在 20 世纪 90 年代自行开发的编程工具——BASIC 的变种 Visual Basic。Windows 的最初版本于 1985 年投放市场，但是直到 1991 年，这种图形操作系统才成为市场上的大赢家。虽然微软已经改进了 Windows，但 1991 年它同时推出了 Visual Basic，这使得在 Windows 上运行的程序更加易于编写。比尔·盖茨评价说：“多年以来，BASIC 一直是我们成功的法宝。”[4]

开发 BASIC 以及达特茅斯大学的分时系统原本都只是为了教学和文化发展。20 世纪 60 年代初期，计算机显然已不仅仅是大型计算器，而是能够为科学、商业领域及政府事务服务的数据和图形处理器。1959 年，C. P. 斯诺在剑桥大学做了一场名为“两种文化”的演讲，深入剖析了理科生与文科生的不同，并且指出了这种差异的危险性。凯默尼和库尔兹都深受这个观点的影响。库尔兹曾评价这两种人为“知道系统如何运行的人”和“仅仅对系统做出回应的人”。他和凯默尼都认为，作为一所常春藤名校，达特茅斯大学在这个时刻要有所作为。因为最有可能对计算机产生兴趣的理工科专业学生仅占全院总学生的 25%，但是，很显然，未来的商业领袖及政府决策者大多会来自剩下的那 75%，那些对技术不太感兴趣的学生。1978 年，库尔兹在他的一篇文章中写道：“我们一直很奇怪，那些对计算机工作一无所知的人如何能对计算机及其应用作出正确的决策？对这一问题得出的结论就是，非理科生也必须学习使用计算机。”[5]

很早之前，达特茅斯大学就提出文科生必须学习计算机课程的理念，同时这一观点也得到了常春藤联盟院校其他教育工作者的赞同和拥护。例如，1979年，全美顶尖计算机专业学院——卡耐基梅隆大学计算机科学学院的院长约瑟夫・特劳布[6]前往哥伦比亚大学，帮助其建立计算机科学学院。约瑟夫回忆这段往事时说道："在那里，我的任务就是要说服这所全美顶尖的综合大学，告诉他们计算机科学的重要性。"他确实做到了这一点。布莱恩・科尼汉曾经是贝尔实验室 Unix 小组的研究员，时至今日，他还在普林斯顿大学教大一新生计算机课程。当回忆起 20 世纪 60 年代在达特茅斯大学的情景时，他说："国家的政策制定者必须明白，计算机已经深入到人们生活的各个角落。如果你不和计算机面对面交流，就不会真正明白在给计算机下指令时需要做得多么精准才能让它明白你的需求，同时你也不会明白哪些东西是会出错的。"[7]他微笑着接着说。因此，在课堂上，"我会让小淘气们编程序"。当然，他说的是用 Visual Basic。

1962 年，库尔兹找到当时任数学系主任的凯默尼。凯默尼还记得当时库尔兹提出了一个"大胆的建议"[8]，那就是让达特茅斯大学的每位学生都学习如何使用计算机。对于凯默尼来说，这个提议并不现实，因为当时计算机使用的还是批量处理的穿孔卡。但是，凯默尼十分认同计算机的重要性。约翰・凯默尼既是一位数学家，又是一位哲学家，后来成为了达特茅斯大学的校长。他出生于布达佩斯，后随家人移居到纽约[9]，高中毕业以后考入普林斯顿大学攻读数学专业。1943 年，这个年轻的大三学生被选派到洛斯阿拉莫斯参与曼哈顿项目。他被分配到这个项目的计算中心，为原子弹的设计开发做计算工作，和其他人一样，每天工作 8 小时，三班倒。

凯默尼于 1992 年离世。如果现在他还在世的话，也许会开玩笑地说，一个达特茅斯大学的大二学生使用学校 20 世纪 60 年代的分时系统，一个下午就能完成一个 20 人的工作团队在洛斯阿拉莫斯一年的工作量。战后，凯默尼回到普林斯顿大学攻读博士学位，毕业后在艾尔伯特・爱因斯坦的高级研究所工

作，担任爱因斯坦的研究助理。凯默尼回忆说，爱因斯坦曾说过，计算机的真正价值在于它已由大型数字计算器发展为图形处理器。凯默尼对这一观点印象深刻。正是软件和更快的处理速度才使得计算机从操控算术符号发展成为处理其他种类的符号的工具，诸如文字符号、物理现象甚至思想理念。这就是为什么我们能够给计算机编程，让它下象棋，模拟天气模式，并最终揭开人类基因密码之谜。甚至 BASIC 的名字都揭示了计算机发展的新里程——S 代表 symbolic（符号化的）。

然而，如何在大学生中普及计算机知识还是一个难题。当时，麻省理工学院的分时计算机实验尚在进行中，但约翰·麦卡锡却是一位大力提倡者。当库尔兹前往麻省理工学院见麦卡锡时，麦卡锡就毫不犹豫地鼓励他说："你们一定要做分时系统。"深思熟虑之后，库尔兹和凯默尼都表示同意。在达特茅斯大学管理层的支持和国家科学基金的资助下，他们开始设计自己的分时系统。

---

20 世纪 60 年代早期还没有出现供计算机新手使用的编程工具。FORTRAN、COBOL 和 Algol 等高级编程语言比汇编语言简单得多，但主要是为科学、工程或商业等特定领域的专家设计的。它们的语法规则对文科学生来说不够直观。在设计出 BASIC 语言之前，库尔兹和凯默尼曾经尝试过一些教学语言。库尔兹和两个学生设计了一种叫 Scalp（Self Contained Algol Processor，独立 Algol 处理器）的语言。1962 年，凯默尼和一个学生共同设计出了另一种叫做 DOPE（Dartmouth Oversimplified Programming Experiment，达特茅斯超简明编程实验）的教学语言。Scalp 和 DOPE 都试图将编程简单化、标准化，虽然它们都不尽如人意，却为 BASIC 的成功打下了基础。

通过总结个人和学生的计算经验，库尔兹和凯默尼共同制定了 BASIC 的 8 条设计原则。这种新的编程语言对新手来说必须简单易学，并且通用，以便能够编写各种程序。其他原则还包括：新的语言必须能够给出简单易懂的错误提

示信息，必须能够升级及改善性能，且升级换代时不应给新手造成困难。他们还加了一条，即使是对硬件一窍不通的人，也能运用自如。

为了简化布局，BASIC 每一行只有一个指令。每一行都以一个操作语句或命令开始。BASIC 语言共有 14 个命令，包括 LET、READ、DATA、INPUT、GOTO、IF THEN 等。为了进一步简化，库尔兹和凯默尼为打印以及计算小数的位数而设计了默认格式，这样用户就不用再为这些细节编程了。每一行 BASIC 编码都有数字编号。这种编号使得用户在改正错误或者转换程序时可以使用相同的行号，或者在删去一行指令时只保留其行号。以下是凯默尼和库尔兹在《BASIC 编程》（*BASIC Programming*）一书中提到的一个例子，是一个简单的算术运算：147 除以 69。

```
DIVIDE
100 READ N,D
200 DATA 147,69
250
300 LET Q = N/D
400 PRINT N,D,Q
450
500 END
RUN
```

在这个简单的例子中，BASIC 程序 DIVIDE，告诉计算机分别读取分子（N）147 和分母（D）69。LET 行的指令为计算公式，告诉计算机商数为分子除以分母。接着告诉计算机通过 PRINT 显示计算结果，并以 END 告知计算结束。计算机自动跳过空白行。编程者特意留第 250 行和第 450 行为空白行，是为了让程序看起来更加清晰易懂。编程者只需输入 RUN 指令，就能命令计算机执行程序。库尔兹后来在达特茅斯大学说，编程都是“在电传打字机的黄色卷纸上完成的”。

计算机做出如下答复：

```
DIVIDE 02 MAY 79 17:53
147 69          2.13043
0.114 CRU 0.055 SECS
READY
```

计算机做出反应，将所需信息，即分子、分母和商数 2.130 43 打印在黄色卷纸上。计算机在给出数据的同时也显示出了时间：1979 年 5 月 2 日下午 6 时左右。这些信息在 1980 年凯默尼和库尔兹著作的文章中都有所记录。这个分时计算机同时还告知用户运行程序所耗费的计算机资源的数量和计算机完成工作所用的秒数。当计算机准备好接收下一条指令时，就会显示 READY。

达特茅斯的分时系统于 1964 年 5 月 1 日正式启用。凌晨 4 点钟，该系统用 BASIC 执行了一个简单的程序，遵循了计算机夜间工作的传统。1964 年秋，库尔兹和凯默尼开始教授大学新生的计算机课程。第一学期，他们用 3 节课向学生们介绍了 BASIC 语言，然后，每位学生有 30 分钟的时间使用电传打字机，这些电传打字机都连接了分时系统。他们要求每个学生编程，并检查出 4 个错误。到了第二学期，库尔兹和凯默尼对教学大纲做了一些修改：每个学生每周有 45 分钟的上机时间。“我们高估了学生们的打字能力。”库尔兹和凯默尼于 1968 年在《科学》杂志上发表文章时这样写道[10]。他们还削减了课堂讲授时间。“我们发现，两节一小时的课程已经足以让新手基本掌握 BASIC 语言了。在第二个小时结束之后，学生们都已经跃跃欲试地想开始编写第一个程序了。”

当然，他们不想每年对数千个学生的编程作业进行一一检查。为了解决这个问题，库尔兹和凯默尼又编写一个小程序。当学生键入 TEST 后，软件会接管学生的程序，判定其通过或者提示如何进行检错。凯默尼和库尔兹在 1968 年这样写道：“这种方式大有裨益，只有学生们知道在程序被接受之前自己犯了多少愚蠢的错误。”

达特茅斯大学的计算机课程在文科生中大受欢迎。计算机编程课程虽然不是必修课，但仍然有 80%的学生选修。与之前计算机穿孔卡和批量处理的形式

相比，分时系统更具有互动性，而这也正是它极具吸引力的一大特性。以今天的标准来看，通用电气的分时计算机功能其实并不强大，而是有很多缺陷。凯默尼和库尔兹评价说："我们发现，如果计算机平均反应时间超过 10 秒，就会破坏人们拥有自己的计算机的幻想。"对于 20 世纪 60 年代那些坐在电传打字机前的达特茅斯学生来说，计算机就好像活的一样，它能够对每个人键入的指令分别做出回应。20 世纪 60 年代至 70 年代初的那段时间，BASIC 让各个年龄段的学生都有种"个人计算"的体验。

---

20 世纪 70 年代中期，由于微型计算机的面世，个人计算这个概念突然由分时系统造成的幻觉变成了拥有自己的个人计算机的现实。分时系统计算机已经过时，BASIC 却顺应潮流而转变。虽然起初使用时还有些蹩脚，但它成为微型计算机革命中编程语言的首选。斯坦福大学的讲师丹尼斯 · 艾利森[11]在他的朋友鲍勃 · 阿尔布莱特的鼓励下编写了早期微型计算机编程语言 Tiny BASIC，即微型 BASIC。鲍勃 · 阿尔布莱特是位计算机工程师，因不满计算机行业只重视为机构和公司服务而不为个人服务，于 20 世纪 60 年代离开了数据控制系统有限公司。随后，他来到旧金山，并很快成为北加州另类的计算文化中心。他创立了小报《大众计算机伴侣》（*People's Computer Company*），专门向普通大众传播计算机知识。由于小报出版后受到读者热捧，阿尔布莱特索性建立了一个叫作 PCC 的非营利组织来运营。1970 年，该组织在门洛帕克的一家购物中心开设了面向普通民众的计算机中心，让一般民众也能用上分时计算机，此时微型机时代尚未到来。

阿尔布莱特与丹尼斯 · 艾利森志趣相投，因此丹尼斯也成为了该非营利性计算机中心的常客。计算机中心有一些键盘终端连接到数字设备公司的一台 PDP-8 小型机上，之后又增加了一些，因为借助电话线连接，惠普公司捐了一些机时。走进这家计算机中心的人形形色色，有小学生，也有成人。使用计算

机的价格优惠，每小时仅需 25 美分。客人到这里的需求也各不相同：解决数学难题、编写小企业商用程序以及体验各式计算机游戏等，从小游戏 Stars and Snark 到复杂的文字冒险类或角色扮演类游戏 Wompit 和 Hummarabi。所有这一切都是基于 BASIC 完成的。PCC 的计算机门店同时也是一群计算机发烧友的据点。他们在这里给孩子过生日，还定期举办自带饭食的晚餐会。PCC 是这些计算机爱好者的乐土，他们视计算机和代码为解放性工具。其成员中有提倡言论自由的社会自由主义者，也有反政府、反企业和反越战人士。《计算机解放》一书的作者泰德 · 纳尔逊也经常来这里参加晚餐会。他自行出版的这本书吹响了计算机革命的号角："你们现在就能而且必须了解计算机!"

坐在斯坦福大学校园里一家寿司店门口的午餐桌旁，艾利森回忆说："PCC 产生的社会影响令人难以置信，它以多种方式引领了个人计算机运动的社会意识。"时代在发展，人的想法也在改变，但是艾利森却坚信"开放、分享、直言不讳"是计算机文化的永恒价值。微型 BASIC 就是在这种精神的鼓舞下产生并传播的。MITS（Micro Instrumentation and Telemetry Systems，美国微仪系统家用电子公司）开发的 Altair 微型计算机于 1975 年 1 月首次出现在《大众电子》杂志的封面上，售价为 400 美元。鲍勃 · 阿尔布莱特看出了这款在英特尔 8080 微处理器上运行的计算机的潜力，认为它是一款大家买得起、多功能的计算机。实际上，当时的 Altair 微型计算机用的只是闭合电路，没有软件，不具备娱乐性和实用性。阿尔布莱特鼓励艾利森开发一种更为简易的 BASIC 版本，以供儿童使用 Altair 和刚刚出现的其他微型计算机。

于是，艾利森开始工作，他写的一篇关于微型 BASIC 的文章刊登在 PCC 和另外一个姊妹刊物《多布博士》上（*Dr. Dobb's Journal*，多布是丹尼斯 · 艾利森和鲍勃 · 阿尔布莱特两个人名字的首字母缩写）。"微型 BASIC 是 PCC 的一次尝试，我们希望为计算机爱好者提供一种更为人性化的语言来编写程序。"艾利森写道。和 PCC 的信条一致，微型 BASIC 被认为是早期计算机教育和计算机改装必不可少的工具。在关于 PCC 的一篇新闻通讯中，艾利森为微型

BASIC 大作宣传。

> 假设你是一个 7 岁的孩子，你当然不在乎什么浮点运算、对数、正弦、矩阵求逆、核反应计算，或是其他类似的玩意儿。你家的计算机有点小，也没多少内存，也许就是一台不足 4 KB 的 Mark-8 或 Altair 8800，或是一台可输入输出的电视电传打字机。
>
> 你想用它来做家庭作业，解数学难题，玩 NUMBER、STARS、TRAP、HURKLE、SNARK 或者 BAGELS 等游戏。
>
> 那不如考虑用微型 BASIC 吧！

艾利森设计出微型 BASIC 并不断改进和完善，使之成为一种用途越来越广的编程语言，不只是 7 岁的孩子，就连真正的计算机爱好者都在使用。微型 BASIC 简单易学，让千百万的程序员开始体验微型计算机的乐趣。有了微型 BASIC，编程者可以随心所欲，不受限制。艾利森和阿尔布莱特在当初发布微型 BASIC 时就决定让其使用者自由发挥，他们从未想过要借此谋利，因为他们都是言论自由和自由软件的坚定支持者。

而在这个国家遥远的另一端，马萨诸塞州的剑桥市，两个创业的年轻人，保罗 · 艾伦和比尔 · 盖茨，却选择了和他们截然不同的道路。他们的传奇故事已经家喻户晓：1975 年，两个年轻人在读过 1 月出版的《大众电子》杂志后，随即着手为 MITS Altair 设计商业版的 BASIC。事实上，这并不是他们两人的突发奇想，而是经过数年的市场观察和编程工作得出的结果。“我们的事业说到底都是源于 1971 年在《大众电子》杂志上看到的一篇文章，文中提到了世界上最早的微处理器——英特尔 4004 芯片。”[12]数年后保罗 · 艾伦解释说，“这篇文章让我意识到计算机的价格将会大幅下降，从而进入千家万户。”

保罗·艾伦和比尔·盖茨[13]是西雅图私立湖滨中学的校友。早在那时，两个人就对计算机产生了浓厚的兴趣。上八年级时，盖茨就开始在分时系统中使用 BASIC 编程了。二人甚至还合伙开了一家名叫 Traf-O-Data 的公司，建造了一台计算机，处理高速公路的交通信息。他们曾商讨过为早期的英特尔芯片编写 BASIC 语言，但最终决定等待，直到使用英特尔 8080 芯片的处理引擎 MITS Altair 上市。当时，保罗·艾伦已从华盛顿州立大学退学，在波士顿的霍尼韦尔公司找了一份工作，而比尔·盖茨仍在哈佛大学读书。“比尔和我都急于建立自己的公司，”艾伦回忆说，“我们都意识到要当机立断，否则就会永远失去这个进入微型计算机软件行业的宝贵机会。”

虽然 Altair 已经宣布上市，却还未普及，但这时的艾伦和盖茨已经迫不及待了。于是，艾伦模拟 Altair 的英特尔 8080 微处理器编写了一个程序以及大部分软件开发工具，然后盖茨用哈佛大学计算机中心的一台 DEC PDP-10 小型计算机，为他们最初的 BASIC 编写了大部分代码，并由另一名哈佛学生蒙特·大卫多夫帮忙编写数学处理例程。盖茨没完没了地使用计算机中心的机器，最终带来了麻烦，尤其是他还带艾伦（他不是哈佛的学生）进来，而且他们做的还是个商业项目。盖茨本来有可能受到学校警告甚至开除的处分，但幸运的是，最终此事不了了之。多年以后，盖茨为母校捐赠了几百万美元，为当年违反计算中心的纪律作出补偿绰绰有余。

盖茨为 Altair 编写的 BASIC 具有很多功能。它的速度很快，而且作为一种最初运行在仅有 4 KB 的内存板上的编程语言（2001 年经济型个人计算机的内存为 64 MB，是其 1.6 万倍），它有很多令人印象深刻的特性。高中时，盖茨曾为小型计算机编写过一个 BASIC 解释器。“那个程序里面错误百出。”[14]盖茨曾经回忆说。但是，这段经历促使盖茨仔细研究其他程序，以便将用 BASIC 语言编写的软件转换或解释成计算机能够执行的代码。在 1975 年盖茨开发出 Microsoft BASIC 之前，他和艾伦已经仔细检查过无数个版本的 BASIC 了。那时问题的关键已不在于盖茨能否写出一个 BASIC 的新版本，而在于他能否让

BASIC 在区区 4 KB 的内存中尽可能快速灵活地运行。“事实上，能让 BASIC 在如此小的内存中运行简直就是个壮举，”盖茨在 2001 年回忆说，“在我编写的所有程序中，BASIC 最让我自豪。”[15]

为了给微型计算机开发出更加强大的语言，盖茨和艾伦并没有遵循达特茅斯 BASIC 的规则，而是将多条编程指令合并到一行代码中以节省空间。而为了更好地控制计算机，他们把所谓的 PEEK 和 POKE 指令也加入其中，这样编程者就可以预览，继而直接操控内存中的数据字节。这与凯默尼和库尔兹当初提出的高级编程语言应简化编程细节的原则背道而驰。微软为 Altair 设计的 BASIC 是达特茅斯 BASIC 与更高性能的完美结合，而大部分改进最初都是在别处完成的。例如，早在 1971 年，美国数字设备公司的程序员就已经把 PEEK 和 POKE 嵌入到一台小型计算机分时系统的 BASIC 程序里面了[16]，但是，盖茨强调，在微型计算机里面加入 PEEK 和 POKE 指令仍然是一项巨大的挑战。他说，美国数字设备公司的小型计算机在当时算是便宜的，但依然价值 5 万美元，“而市场上需要的是仅需 500 美元的大众计算机，因此我们即刻动手了”。

像 Microsoft BASIC 这样将速度与性能完美结合的小程序确实令人惊叹。而凯默尼和库尔兹将他们专门为微型计算机编写的BASIC称作“街头BASIC”。尽管他们勉强对盖茨和艾伦于 1975 年的工作成果表示赞赏，称之为“引人注目的成就”，但也没忘了加上一句“对 BASIC 语言来说却是灾难”。现在回过头来看，库尔兹承认早期微型计算机的硬件制约太苛刻，但仍然认为 Microsoft BASIC “失去了达特茅斯版 BASIC 的简洁”。虽然 BASIC 在从教学语言发展到工作语言的道路上历经坎坷，但最终还是在现代个人计算机产业的发展中留下了浓墨重彩的一笔。

1975 年春天，盖茨和艾伦与艾德 · 罗伯茨签下一份合同。艾德 · 罗伯茨是位于新墨西哥州阿布奎基的 MITS 公司的总裁。在合同中，两人称自己为“微–

软（Micro-Soft）的保罗·艾伦和比尔·盖茨”。经过长时间的讨论，公司的名字最终成了现在的模样：组合了“微型计算机”（microcomputer）和“软件”（software）这两个词的前半部分。这份合同对盖茨和艾伦来说至关重要，因为MITS公司开发的Altair计算机是当时羽翼未丰的微型计算机行业的领头羊。而此时的盖茨虽然已从哈佛退学，却仍然野心勃勃。他并不完全依赖MITS公司，而是保持一定的独立性，要确保与该公司签订的合同不能限制他自由地将Microsoft BASIC出售给其他微型计算机制造商。盖茨希望与尽可能多的计算机制造商签约，从而使Microsoft BASIC成为行业标准。至于销售的对象，他坚信，与其零售给业余爱好者，不如批发给制造商，这样更赚钱，至少可以把钱收回来。“计算机爱好者喜欢我们的BASIC，”盖茨在25年后回忆道，但他同时也强调，“他们只想从别人手里‘借’来用，觉得我们还要收钱太不公平了。”[17]

1976年，盖茨对于“软件盗版”问题忍无可忍，便在该年2月3日以嘲讽的语气向计算机爱好者发出了一封著名的“公开信”，这封信发表在美国硅谷计算机发烧友聚集地“家酿计算机俱乐部”的时事通讯上。“计算机爱好者必须明白，”他写道，“你们中大多数人使用的是偷来的软件。硬件必须要付款购买，可软件却变成了某种共享的东西。谁会关心开发软件的人是否拿到了报酬？”他接着说，他将“感激那些愿意付款的人的来信”。

盖茨的这封信并没有带来潮水般的支票，对此他并不感到意外。他表达了自己的观点，但已经把重点转为向个人计算机制造商推销他的BASIC。盖茨虽然年轻，却有着敏锐的商业嗅觉，甚至还作为推销员去推销这款软件。他来到位于得克萨斯州的拉迪奥沙克公司，向他们推销可用在TRS-80个人计算机上的BASIC软件。在那里，他见到了当时的副总裁，也就是后来的董事长约翰·罗奇。听到盖茨5万美元的开价，罗奇的回答是：“扯淡！”盖茨差点儿被这种得克萨斯式的讨价还价吓倒。但是，后来他回忆道：“我坚持不让步，争辩说这款软件正是拉迪奥沙克公司的客户需要的，是他们购买的个人计算机中

至关重要的一部分。罗奇是个很难对付的家伙，但他最后还是接受了我的报价。"一次次的成功推销让盖茨信心大增。"当来到德州仪器公司时，我决定开价 10 万美元，"他说，"不过又怕 6 位数字的报价会吓跑他们，所以我开价 9.9 万美元。结果成交，他们买了。"

Microsoft 的软件绝不是当时市场上唯一的微型计算机版 BASIC。1976 年，加州蒙特利海军研究生院的学生小戈登 · 尤班克斯[18]编写了 CBASIC。尤班克斯曾经在位于静水市的俄克拉荷马州立大学就读，早在那时他就已经显示出编程的才能。在进入加州海军研究院深造之前，他曾在图尔萨的 IBM 公司工作过 6 个月。期间，他为壳牌石油公司编写程序，查找全国范围内的壳牌加油站的加油卡是否有欠款记录，并自动向有不良记录的用户发送催账单。"这种令人惊叹的新技术能够解决实际问题，"尤班克斯说，"我真切地感受到了那种力量，而且发现它绝对让人着迷。"尤班克斯的 CBASIC 仅售 100 美元，在一小部分以编写特定商业软件为生的半专业人士中大受欢迎。与此相反，盖茨的着眼点落在利润丰厚的个人计算机制造商身上，他与这些制造商密切合作，说服他们把软件开发交给微软。"我们对每一家硬件制造商的需求都非常敏感，"他说，"我们不想给其他人任何机会，我们希望微软成为不二选择。"

多年以来，尤班克斯与微软有竞争也有合作。尤班克斯理解为何微软在如此短的时间里就成了软件行业的巨头。"比尔一直很有商业头脑。"尤班克斯评价说。

对于尤班克斯而言，CBASIC 是个夜间项目，每天工作到半夜 1 点左右才能睡觉，早上还要按时起床去海军研究院学习。在这个夜间项目上，他有一个助手——阿兰 · 库珀，库珀做了很多事情，其中包括为 CBASIC 编写用户手册。"如果没有库珀的帮助，"尤班克斯说，"我永远都不可能完成 CBASIC。"库珀及其合作伙伴基斯 · 帕克森帮助尤班克斯是有原因的。尤班克斯开发的 CBASIC

有中间层代码，这使得用这种语言编写出来的程序很难被复制、修改或剽窃。这对库珀和帕克森来说尤为关键，因为他们合开了一家公司，推出的第一款产品便是通用账簿程序，这是最早为微型计算机开发的商务软件之一。库珀回忆说，他的目标是有一天能够实现 5 万美元的年销售额。

库珀是个人计算机方面的业余发明者。1969 年高中辍学以后，他开始了漫无目的的旅行，从加州、阿拉斯加到西欧，到处都留下了他的足迹。用他自己的话说，这是一个“严肃的、自我疗伤的过程”。从当时的一张照片上可以看到，库珀长发披肩，微笑着站在一辆粉刷着怪异图案的面包车旁边。到 1972 年，他厌倦了这种四处游荡的生活，重新读完高中后，考入旧金山北部马林县的一所社区大学就读。他本想学习建筑学专业，却阴差阳错地选择了计算机编程。

“刚学了几个星期，我就上瘾了。”库珀在他帕洛阿尔托的办公室里回忆说。当年的嬉皮士如今已是一家软件咨询公司的老板。“我开始狼吞虎咽地学习计算机和编程”，不放过任何一节数据处理课程，还在学校的计算机中心工作。“编程是完美的可控媒介，”他解释说，“你可以用代码做你想做的任何事情，你有绝对的控制权……我再也不想去学建筑学。”就这样，当微型计算机革命到来之时，库珀立即投身其中，开始时与基斯・帕克森合作，后来自己单干。库珀的工作风格是在他自己的软件工作台上埋头苦干，开发新软件或者解决些技术难题。“我越干越上瘾，完成了一个又一个的项目。”他说。他找到的项目足以谋生，但仅此而已。

20 世纪 80 年代后期，库珀针对所谓的“高手”用户推出了一款灵巧的工具。所谓的“高手”用户，是一群专注投入的业余爱好者，他们喜欢用自己的电脑慢条斯理地工作，却并非计算机专业人士。这款名为 Ruby 的程序是一种“壳结构集”。它为用户在屏幕上展现了一个长方形的“面板”，类似于工作台，用户可以在上面修改操作系统的图形外壳。有了 Ruby，用户就可以用鼠标将几个项目拖拽并放置在面板上，再将其组合起来。“这有点像数字积木。”库珀

解释说。Ruby 能够让技术娴熟的用户对程序进行微调和修改，或者将两种工具组合起来。Ruby 的桌面“面板”可以充当用户喜欢的应用程序的发射台。

库珀很快认识到，真正能让他的发明有用武之地的只有微软公司。1988 年 2 月，库珀来到位于西雅图郊区的微软公司，向盖茨的一位年轻得力干将加布・纽维尔展示他的原型软件。预计 5 分钟的演示结果持续了 1 个小时，最后，纽维尔抬手打断了库珀，说道：“我一定得让比尔看看这个。”与盖茨的会面安排在了下个月。库珀说，在这一个月中，“我就好像疯了似的，不断改进我的程序”。盖茨确实被打动了，长时间的演示终于结束，盖茨当场就对他的团队说：“我们要向这方面发展。”库珀最终决定不加入微软公司，而是将 Ruby 卖给了盖茨。“如果当时我加入了微软，我早在帕洛阿尔托赚大钱了。”这位业余发明家指出，“不过我一点也不后悔。”

盖茨最初的想法是把 Ruby 囊括到 1990 年上市的微软 Windows 3.0 操作系统中。自 1985 年第一代面世以来，Windows 的道路并不平坦。毋庸置疑，DOS 操作系统一直以来都是这家公司的一棵摇钱树。但是，DOS 的命令行界面已经过时，用户不希望再通过键入文本指令来打开应用软件或显示文件列表。DOS 称霸市场的日子眼看就要结束了。最终，微软决定放弃在 Windows 中加入只有老练的用户才能运用自如的 Ruby。当时，微软的图形操作系统在简单易学和用户体验方面还比不上苹果公司的 Macintosh，而像 Ruby 这样不招人待见的工具可能也无助于 Windows 获胜。盖茨回忆，当时库珀和微软内部的一些人都极力争取让 Ruby 成为 Windows 系统中的“壳定制工具”，但是，他说：“那是死路一条。人们并不希望用这种方式定制外壳。”

实际上，微软并没有完全舍弃 Ruby，而是让它以编程工具的形式获得新生。Windows 面临的一大问题在于，为微软图形操作系统编写应用程序的那些企业进展太慢了。而且当时 C 语言已经出现，并取代 BASIC 成为最常用的个

人计算机编程语言，这让微软的处境雪上加霜。但是，C 语言出自贝尔实验室的计算机科学家之手，肯定是一种最适合真正的专业人士使用的语言，尤其是适合编写复杂的图形应用软件。因此，当时能够用 C 语言编写 Windows 应用程序的人简直就是凤毛麟角。无论如何，微软必须想方设法让业界更多的人为 Windows 编写程序。盖茨认为让 Ruby 和 BASIC 联姻可以做到这一点。Ruby 是一种可视工具，可将其点击式的特点应用到具有增强功能的编程语言——微软的 Quick BASIC 4.0 上。

将这二者合二为一并非易事[19]，但是经过一年半的艰苦努力，Visual Basic 终于在 1991 年 5 月横空出世。Windows 成为行业内的垄断性操作系统，Visual Basic 功不可没。它使得个人计算机能够向商业计算领域进军，处理新任务，还能处理原先只在大型机或小型计算机上才能完成的工作。而用 Visual Basic 编写商业应用程序，只需在屏幕上拖拽图标，然后合并编码就行了。

经过一两天的培训，公司技术部门所有的 COBOL 和 BASIC 程序员就都学会了使用 Visual Basic 编程。公司的程序员大都负责编写特定的商业程序。他们通常 1~5 人为一组，为了完成同一个项目共同工作几个星期到几个月。这些程序员开发的并不是数百万人使用的操作系统，而是为公司内部为节约资金或增加销售而定制的实用程序。这些程序可用于监控工厂车间的机器运转，评估广告宣传效果等。Visual Basic 为庞大而沉默的公司程序员群体开启了 Windows 编程之窗。

看着 Visual Basic 的产品，阿兰・库珀为当年卖给微软公司的软件能有今天的成功感到无比惊喜。“微软的工作团队干得太棒了，但我还是觉得十分惊讶。”库珀说，“这种感觉就好像把你的孩子送去上大学，几年后他衣锦还乡。他的变化太大了，你要好一会才能适应。”当时库珀以 100 万美元的价格将 Ruby 出售给微软，并赢得了“Visual Basic 之父”的美誉。（微软的律师曾向库珀发出禁止令，要求他停止使用该名号，但库珀据理力争，因此，盖茨在一次行业会议上承认库珀所作的贡献，并称赞其为“Windows 的先驱”。）回首往事，最

让库珀心服口服的是盖茨的战略眼光。“我知道我做的软件很酷，”库珀说，“但我从来没想过它能成为编程控制面板，并为微软带来今天的成就……比尔·盖茨就是比尔·盖茨啊！”

比尔·盖茨并不是一位优秀的技术创新者或伟大的程序员。他设计的 BASIC 却是一个例外，与其说它是真正的创新，不如说它是杰出的工程成就，尽管二者对软件来说是很细微的差别。但是，最重要的是，盖茨对软件行业有着深刻的理解，同时还是过去半个世纪中最优秀的应用经济学家。20 世纪 80 年代初，一群年轻的经济学家就开始研究技术市场的行为，以及行业标准、兼容性产品和培训成本对竞争的影响。他们提出了“网络效应”和“转换成本”的概念，当时，大型机计算时代的 IBM 公司和电话系统都是他们实际研究的对象。

然而，这些经济因素都受到软件行业内的特殊因素制约。对于复杂的技术产品来说，用户的“转换成本”很高，但是，一旦程序被开发出来，软件产品的成本（仅仅是复制）就几近为零。因此，软件行业的用户更愿意使用已经普及的行业标准，而如果行业标准仅为一家公司所有，那么它将为这家公司在胜者为王的市场中带来高额利润。迈克尔·卡兹和卡尔·夏皮罗在 1985 年发表的“网络外部性、竞争性及兼容性”则是这一新型经济研究领域中具有开创性的学术论文。据夏皮罗回忆[20]，他们写这篇文章时，卡兹告诉他，自己还在读大学时，就已经有人在西雅图郊区一家软件公司工作，所做的事情正如他们的论文所写的那样：引领软件行业的标准，稳操市场胜券。而且这个人曾在哈佛大学就读。没错，就是比尔·盖茨。

利用 BASIC 可塑性极强的特点，盖茨推出了他的 Microsoft BASIC，充分发挥了软件的经济性。虽然 BASIC 的发展已经脱离了达特茅斯大学那两位教授设计这种教学用编程语言的初衷，但它的命运在早期个人计算机迅猛发展的

年代就已注定。阿兰・库珀曾直言不讳地说："BASIC 是一种'逆来顺受'的语言，任何人都可以按照自己的想法随意摆弄它，为自己的商业或技术目的服务。" 20 世纪 90 年代后期，微软已成为市场的主宰，而它因为其在软件市场上的行为违反了美国反垄断法案，把自己推上了联邦法院的被告席。2001 年 6 月，联邦法院裁定，微软公司的一些商业行为属于非法垄断，恶意压制来自网景通讯公司的商业竞争，这是一家早期的互联网搜索软件公司，但上诉法院驳回了拆分微软的裁定。这都得益于早在公司将 BASIC 打造成行业标准时，盖茨对软件经济超乎常人的真知灼见，才使得微软公司成为大赢家。

一些程序员打趣地说，Visual Basic 唯一保留原 BASIC 的东西恐怕就剩这 5 个字母了。似乎为了表明一切都有所变化，微软在 Visual Basic 这一名字当中也不再使用五个大写字母了。"我们充分利用了 BASIC 语言可塑性强的特点。"[21] 负责开发工具的副总裁汤姆・伯顿这样说道。微软自 1991 年发布 Visual Basic 以来，再一次娴熟地建立了行业技术标准。到 2001 年，Visual Basic 已成为数百万软件开发人员使用的编程标准和首选的编程语言。对微软来说，迎合软件开发人员，拉拢他们，帮助他们，向其提供有用的工具，让他们对微软的技术产生依赖性，是公司的一大使命。微软的格言是："平台就是一个生态系统。"软件开发者是这个系统中至关重要的一部分，微软所做的就是不遗余力地满足他们。

每年，微软都会赞助数百场会议或培训，全球共有百万程序员参加。微软公司中数千名员工的工作就是与外界的软件开发者打交道。"保罗・艾伦和比尔・盖茨在创建微软时的远景规划又回来了，"汤姆・伯顿解释说，"他们最先做的是什么？他们为成千上万的程序员设计了一种编程语言，用以编写软件应用程序。紧紧抓住程序员的需求就是成功的关键。这是计算机行业永恒不变的真理，微软的成功就在于此。"

Visual Basic 和微软公司本身一样，起初都是为了个人计算机产业应运而生的。随着互联网的兴起，个人计算机将在计算机行业中退居二线。微软像业界

其他公司一样，对此心知肚明，而面对这样的挑战，它所做出的回应是试图建立一个叫做.Net 的互联网编程平台。这一策略的重点是要将使用 Visual Basic 的程序员吸引到微软的互联网编程平台上。不出所料，公司推出的 Visual Studio.Net 如同 Visual Basic 一样，又是大获成功。微软向程序员传达的信息是：会用 Visual Basic，就会互联网编程。

2000 年夏天，微软专业软件开发员大会在奥兰多召开，面对 6000 多名程序员，比尔・盖茨强调，未来公司在互联网方面必将复制过往的辉煌。他说，我们搭建软件平台，吸引程序开发员，“这是公司一直以来不变的追求”[22]。

第 6 章

欧洲的影响力：从 Algol 到 Pascal 再到 C++

作为最重要的软件工具，编程语言的类型和结构多种多样；但从 FORTRAN 和 COBOL 到 Visual Basic 和 Java，这些主流编程语言都有一个地理上的共性：诞生在美国。但是，美国绝不是编程语言唯一的发源地，欧洲的发展成就也不容小觑，像 Algol、Simula 和 Pascal 这样的编程语言，虽然没有在商业上大获成功，却有重要的学术意义。美国人似乎在设计编程语言时融入了工程学思想，采取折中的方式解决计算机应用方面的实际问题。相比之下，欧洲人在设计语言方面更偏重学术理论，美国人则更重视经济效应。

出现这种情况的原因比表面上看起来更加微妙：美国是世界上最早发展计算机市场的国家，而“资本主义的动物性”也主要体现在美国。第二次世界大

战后最初的几年，美国的软件发展趋于实用性，这源于多种因素，其中政府的支持扮演了重要角色。当时，美国计算机行业的发展方向与政府扶植和航空航天工业的发展，尤其是军事需要，密不可分。例如，在 20 世纪 50 年代的鼎盛时期，有 7000 多名 IBM 员工为美国国防部的 SAGE 监控项目和防空系统工作，占全公司员工总数的 20%[1]。这些为国防部和航空技术公司工作的工程师和程序员，利用计算机设计战斗机、商用客机和计算机控制的雷达系统。华威大学的计算机历史学家马丁・坎贝尔–凯利评论道："美国人将计算机用在实际工程中，是因为他们认为只要能迅速达到目的，就可以不择手段。"[2]当时的美国既拥有金钱又拥有最多的计算机，因此美国人自然最有可能试图运用这些机器做点实事，应对挑战。而欧洲既没有足够的财力，也没有如此的功利性，所以更从容的理论研究方法才会在那里生根发芽。

无论出于何种原因，美国与欧洲之间的这种差异近几年已日渐模糊。美国依然是全球最大的计算机市场，但已失去昔日的统治地位。与此同时，互联网的兴起使新观念和新软件能够迅速跨越国界，扩展到全球市场。在过去的 10 年中，两项软件发展史上的重大进展就源自欧洲：英国人蒂姆・伯纳斯–李在瑞士发明的万维网，芬兰人林纳斯・托瓦兹开发了大受欢迎的 Linux 操作系统。这两项发明都是计算机用于实际工程的产物，即设计者为了解决眼前的实际问题而发明的新工具。

当时，有关软件开发新思想和新技术的国际交流并没有受到足够的重视。简单回顾一下 C 语言的发展历程，我们就会得到很多启发。C 语言是由贝尔实验室开发的，但它是在 BCPL 语言的基础上演变而来的。BCPL 语言，意为 Basic CPL[3]，是由麻省理工学院的英国学者马丁・理查兹设计的；而 CPL 则是由剑桥大学和伦敦大学合作开发的一种编程语言。在伦敦的设计团队加入之前，"C"代表剑桥大学（Cambridge），后来才正式改为代表"组合"（Combined），但圈内人士都知道，其实"C"代表克里斯托弗（Christopher），因为正是有了克里斯托弗・斯特雷奇才会有 CPL。斯特雷奇后来成为牛津大学计算机实验室

的编程研究负责人。他坚信，只有实践与理论相结合才能产生最大的效益。他曾经写道："我始终认为，将实践与理论分割开来是武断的和有害的。计算机领域中很多的软件和硬件设计都没有牢固的理论基础，因为设计者对设计的基本原则没有清晰的认识。而大多数抽象的数学计算和理论工作最终也没能结出硕果，因为这些工作没考虑到实际应用。"[4]

斯特雷奇于 1975 年离世。如果他还在世的话，一定会为本贾尼・斯特劳斯特卢普感到自豪，这个在美国工作和生活的丹麦人在斯特雷奇的 CPL 基础上设计出了 C++。斯特劳斯特卢普采用了欧洲软件开发的思想，并进行了修改和完善，然后将其深深植入到 C++编程语言中。斯特劳斯特卢普一直致力于将理论与实践相结合。他曾就读于剑桥大学，获得博士学位之后，他的职业生涯大多是在 AT&T（美国电报电话公司）的研究实验室里度过的。斯特劳斯特卢普说，C++语言受到了哲学家亚里士多德和基尔克果，以及文学家阿尔伯特・加缪和乔治・奥威尔的极大影响[5]。然而，他并不是一个夸夸其谈、闭门造车的理论空想家。

斯特劳斯特卢普首次接触计算机时还是一个在丹麦上学的大学生。虽然他觉得那些编程理论证明题十分枯燥无味，但在计算机方面的天赋使他比其他学生技高一筹。大学二年级时，他发现计算机编程能给他带来商机，于是很快找了一份兼职工作，在丹麦第二大城市——波罗的海港口奥尔胡斯——为一些小企业编写商用程序。

斯特劳斯特卢普早期在丹麦的工作经验很大程度上决定了他日后在计算机领域的发展方向。后来，他在宝来公司的丹麦办事处工作；这是一家专做台式商用计算机的美国公司。在宝来做程序员时，他经常走访当地企业，倾听它们的需求，然后编写程序以解决它们的问题，这是一种类似于定制编程的服务。他为木材厂、借贷公司、墓碑制造商等各类客户编写财务、记账及工资支付等程序。为了节约宝来公司有限的计算机资源，这些工作都是他用汇编代码手工完成的。

“我觉得自己懂些编程，”斯特劳斯特卢普回忆说，“我知道的并不多，但足以让我赚些钱，并学到一些东西。我能做的就是比别人多往机器里塞些东西。”

渐渐地，斯特劳斯特卢普在编程方面熟中生巧，从之前的项目中吸取精华，加以改进，再用到新的程序中。例如，他为墓碑制造商设计的财务系统就是由之前为木材厂提供的程序改编而来的。他在设计抵押贷款计算程序方面尤为在行。到了 1975 年，斯特劳斯特卢普获得硕士学位时，全丹麦 1/4 以上的借贷公司都在使用他设计的软件。20 世纪 70 年代初，在丹麦与客户面对面交流的兼职工作对他产生了深远的影响。他明显地感觉到，数学和理论性的学术研究与现实世界的编程有很大差别。斯特劳斯特卢普在他新泽西州郊外临湖的办公室中回忆道：“我喜欢设计东西并看到它们发挥作用。你能从理论证明中学到知识，但这远远不够。软件是设计出来的，要为解决实际问题服务。能够发挥作用，改善人们的生活，这才至关重要……帮助大多数人解决他们面临的问题最为实际，而看到积极的反馈又使我产生更大的动力。”

在丹麦工作的那些年，斯特劳斯特卢普培养了注重实际的工作作风，并在后来的工作中一直保持了下去。他的家庭教育也培养了他务实的心态。斯特劳斯特卢普说自己是“工人阶级出身”，因为他的父亲依贡是个娴熟的家具装垫工，而他的母亲英格丽则是个秘书，算是“家里的知识分子”。作为一名计算机科学家，斯特劳斯特卢普可以说是个“工人阶级出身的知识分子”。他是一位与代码打交道，同时又追求实际的哲人。他发明的编程语言 C++继承了欧洲在理论、设计和结构方面的优秀传统，同时又不乏美国人解决实际问题、注重效率和经济效益的特点。

---

50 岁的斯特劳斯特卢普一直坚持长跑和徒步旅行，他身材清瘦，穿着计算机研究人员典型的休闲服装：牛仔裤、跑鞋和衬衫，不系领带。这是 2001 年

6

年初的一天，他坐在位于新泽西州弗朗汉姆公园的办公室里。这里是 AT&T 实验室的大本营（1995 年 AT&T 拆分时，该实验室作为贝尔实验室的一部分留了下来，贝尔实验室的另一部分则跟随朗讯科技公司离开）。斯特劳斯特卢普留着短短的胡子，还有点冷幽默。他博览群书，涉猎广泛，从严肃的历史、哲学书籍到道格拉斯·亚当斯的《银河系漫游指南》和雷蒙德·钱德勒的《漫长的告别》都是他的最爱。

斯特劳斯特卢普的 C++语言是 Unix 大家族中的一员，从丹尼斯·里奇为 Unix 设计的 C 语言发展而来。C++语言其实是 Unix 研究项目的副产品。这个项目的主要目的是将 Unix 操作系统用在相互连接的小型计算机网络中。该项目是对分布式计算的早期尝试，长期目标是把众多小型计算机连接起来，进而建立一个强大、灵活而又相对低廉的计算环境。1979 年，斯特劳斯特卢普在加入贝尔实验室后不久，就开始为该项目做研究。他很快发现，想要拆分并分发 Unix 模块给多个计算机，首先需要借助一种模拟工具来分析网络流量。于是，斯特劳斯特卢普开始着手模拟工具软件的开发工作。

结果这成了一项浩大的工程，4 年之后，一种新的语言 C++问世了。“我从没有真正地开始那个 Unix 项目，”斯特劳斯特卢普笑着回忆说，“我的精力全部用在开发工具上了。”这种模式在软件行业可以说是屡见不鲜。一个本来用于解决某一问题的工具却成了通用的解决方案。这种工具本身自成一体，而最初的项目有时却被抛到脑后了。这就好像一个人想建造一间房屋，却发现需要用到锤子之类有用的工具，于是就发明了锤子；后来他又发现也许有钉子或锯子就更好了。就这样，他最终也没有建造出当初想要的房屋，但他发明的工具却帮助其他人建造了成千上万间房屋。“C++的设计初衷就是要发明一种能够更好地表达思想的工具，”斯特劳斯特卢普说，“而千百万人却对这一系列的新思想产生了兴趣。”

C++语言最终成为了一种功能强大的编程工具。自 1983 年问世以来，C++语言在 20 世纪八九十年代非常适用于编写大型程序，以满足对复杂软件不断

增长的需求。对于程序员来说，C++语言是一种能够组织、定义和处理数据的工具。有了它，程序员能够将一种类型的数据（如员工信息）输入到一种软件存储器中，并对其下达指令；同时在另一种软件存储器中输入其他数据（如科学数据），也对其下达指令，做复杂的数学计算。这些数据存储器，或者说模块，能够合并成相关的模块组。有些存储器能够将信息传递给类似的存储器，而有些则不能。

当程序员似乎要再次失去对软件的控制时，C++语言帮助其在大型程序上理清逻辑和结构。20 世纪 80 年代的计算机功能已十分强大，能够运行大型程序，但是对于软件开发者来说，编写这种程序却变得越来越困难，触发 20 世纪 60 年代末期软件“危机”的困境再次出现。20 世纪 80 年代的大型计算机程序耗资巨大，性能也不稳定，行为难以预测。而 C++语言的出现提供了组织和简化这些大型程序的机制，使程序员能够有选择地使用，并针对其设计的软件制定规则。与其他编程语言不同，C++语言几乎没有一定之规，它提供了一种自助选择方式，批评者认为这会让普通程序员摸不着头脑。

但是，这并没有对 C++语言的传播造成太大阻碍。它被纳入 Unix 体系，并常常用于诸如电话交换机这样重量级的应用程序中。C++语言继承了 C 语言家族的优良传统，让程序员能够在底层对计算机性能进行精准的控制。因此，C++语言逐渐受到业内人士的欢迎，经常用于编写软件以解决棘手的复杂问题，诸如引擎燃料喷射器的控制等。C++语言十分灵活，能应用在不同的机器中，并且经常在恶劣环境下工作。这也正是澳大利亚国家彩票公司将其装在投注机上的原因——大部分投注机都设在日照强烈的内陆地区。斯特劳斯特卢普清楚，在更远的将来，大多数“计算机”将不仅仅是一台有键盘和屏幕的机器。而 20 世纪 90 年代初，随着个人计算机软件公司和其他公司（如宝兰、微软、IBM 和数字设备公司等）开始普及 C++语言，C++语言已登上了现代编程的中心舞台。

很显然，C++语言是斯特劳斯特卢普一个人的成就[6]。在《C++语言的设

计与发展》一书中，斯特劳斯特卢普解释说，这完全是靠他自己的知识，当时也没有其他的计算机技术书籍可供参考。C++语言是在他个人“世界观”的基础上融入计算机科学理念而形成的。几十年来，斯特劳斯特卢普一直坚持阅读历史和哲学类书籍。他说，广泛阅读“让我学会将各类知识融会贯通”。

“在当今的各种思想流派中，”斯特劳斯特卢普写道，“我更喜欢经验主义而非唯心主义，对神秘主义不敢恭维。换句话说，我更喜欢亚里士多德而不是柏拉图，更喜欢休谟而不是笛卡儿……我觉得，克尔凯郭尔对个人主义的狂热和心理学上的敏锐洞察力比黑格尔或马克思充满人文关怀的长篇大论更吸引我。如果忽视群体中存在的个体，那么，所谓对群体的尊重根本就不是真正的尊重。我不喜欢强迫别人以特定的方式做事，这是 C++语言中很多设计理念的来源。”

“因此，” 他后来写道，“C++语言的设计支持百家争鸣而非只听一家之言。”

然而，并不是所有人都能掌握 C++语言，它是为真正的程序员而设计的。斯特劳斯特卢普在 C++语言中表现出来的民主只适用于那些精英人士。也许 C++是一种精英语言，但它的确没有多少对新手的指导。“我希望 C++能使优秀的程序员更上一层楼，而不是保护那些平庸的程序员不犯各种愚蠢的错误，”他在新泽西州的办公室里解释道，“我期盼着专业人士能做些有用的、改善我们生活的设计，如电话交换机、转账系统等。因此，C++不能防止新手犯一些愚蠢的错误。”

斯特劳斯特卢普设计的语言逐渐成为贝尔实验室解决疑难问题的法宝，而这些疑难问题基本上是新手一辈子都不可能遇到的。毫无疑问，这些研究问题大都源于电信领域不断增加的复杂性，因此，斯特劳斯特卢普经常做的工作就是模拟大型网络的数据通信。曾有人请他模拟纽约曼哈顿区的数据通信（从华

尔街的金融区到市中心的公司总部大楼，再到郊区的居民住宅区）。当时，他忠告此人这是不可能完成的任务。他回忆说："但现在回想起来，那时就有人试图寻找联通曼哈顿区的最佳方法了。这项工作非常具有现实意义。"

斯特劳斯特卢普说他"梦想"的编程语言是将欧洲的两种计算机语言 Algol 和 Simula 合二为一。尤其是 Algol 语言[7]，它大致体现了欧洲对软件的贡献，即优秀的创意，但包装和实施的力度不够。Algol 语言（Algorithmic Language，算法语言）出自专业委员会，主要版本 Algol 60 于 1960 年发布，Algol 68（斯特劳斯特卢普最喜欢的版本）于 8 年后推出。Algol 是多国计算机专家，尤其是欧洲大陆的本土科学家共同努力的成果。开发 Algol 的目的就是为建立计算机语言的正式语法规范和逻辑打下基础。

20 世纪 60 年代占主导地位的编程语言是 FORTRAN 和 COBOL，它们都诞生于美国。Algol 委员会成员认为这两种语言就是各种有用特性的大杂烩，其语法对于用户来说看起来"熟悉"；这些用户包括使用 FORTRAN 的科学家和工程师，以及使用 COBOL 的业务经理和会计师。当然，这些 Algol 专家说得很对。通过制定计算机语言的基本规则，Algol 设计小组希望建立起正规的、能够传授的知识体系。他们想把计算机发展成为一门科学、一个学科。作为纯语言设计的一次实验，Algol 60 成功了。十多年后，英国计算机科学家安东尼·霍尔爵士宣称，在语言设计原则方面，Algol 几乎做到了"前无古人，后无来者"。[8]

但是，Algol 的设计者很少与机器打交道，在计算机应用方面缺乏实战经验。曾任剑桥大学计算机实验室主任的莫里斯·威尔克斯指出，1960 年的欧洲大陆上性能强大的计算机很少。因此，Algol 小组并不像他们的美国同行那样，"习惯于每天在计算机中心解决问题。他们对理论更感兴趣[9]……他们决定不被特定机型上的应用问题所影响。如果编程语言的某个特性不能有效发挥作用，就让它顺其自然。"由于缺乏现实应用，Algol 60 一直发展缓慢，没有取得多少进展。

20世纪60年代末期，Algol的普及推广又一次以失败告终。瑞士计算机科学家尼古拉斯・沃斯[10]是 Algol 小组（该小组成立于 1964 年）的一员，他忘不了当时围绕编程语言的基本原则、定义方式和句法细节而展开的“无止境讨论”。Algol 小组很快分成了两个阵营。他说，“激进派成员”希望开拓语言设计的新疆域，“树立起像 Algol 60 一样的里程碑”。但是，像沃斯这样的实用主义者却希望改进 Algol 60，消除其现存的缺陷，并加入有用的特性，包括 COBOL 的一些特性。但是实用主义者失败了。1968 年，具备众多优点的 Algol 68 问世，但它是一种复杂的语言，编译它然后在计算机上运行这事让人头痛不已。因此，即使它在理论上血统纯正，大部分用户仍对其嗤之以鼻，直到 20 世纪 70 年代初才进入市场销售。

这一结果令沃斯十分沮丧，于是他开始着手设计一种与 Algol 十分相似的编程语言——Pascal。Pascal 是以 17 世纪法国哲学家、数学家布莱兹・帕斯卡尔的名字命名的。帕斯卡尔发明了一种计算器，是现代数字计算机的鼻祖。同 Algol 一样，Pascal 也采用了结构化设计，具有代码块，且对不同类型的数据有清晰的定义。和 Algol 相比，1970 年发行的 Pascal 更像是一种实用的工作语言，脱离了纯理论，增加了实用性功能。沃斯在美国的经历显然对他产生了影响。虽然他是在瑞士联邦理工学院设计出的 Pascal 语言，但是他曾在美国接受了计算机教育。他先在加州大学伯克利分校读完了博士课程，后在斯坦福大学教授了 4 年计算机科学课程，然后于 1968 年回到瑞士。

沃斯有着很丰富的计算机操作经验，坚信理论应当服务于实际应用。他说：“我不相信靠讲授的工具和形式能够解决实际问题。”很多重要的程序都是用 Pascal 语言编写的，其中包括苹果公司的 Macintosh。20 世纪 90 年代初，硅谷的软件制造商宝兰公司开发出了 Turbo Pascal。这种在 Pascal 的基础上发展而来的编程语言很快被用于编写在微软 Window 系统上运行的软件，并受到了软件开发者们的一致好评。20 世纪 90 年代末，Pascal 成为大学里讲授结构化编程，尤其是编写严谨的计算机科学程序所用的语言，因此对后世影响颇深。

沃斯于 1975 年面世的《算法+数据结构=程序》一书被认为是计算机科学的经典著作。学习 Pascal 语言的学生与那些学习长期主导个人计算机产业的 BASIC 语言的学生不尽相同。“Pascal 语言能够促使人们从数据结构的角度清晰地思考问题，”[11]沃斯在苏黎世的学生、宝兰公司的创立者菲利普·卡恩说，“使用 BASIC 语言时，你考虑的是带有行号的单个指令，因此 BASIC 程序员不断地跳来跳去，写出的代码就像是意大利面条。”

“沃斯的影响极其深远，”他说，“因为有如此之多的人学习并使用 Pascal 语言编写计算机科学程序。它代表了计算机领域的正统思想。每天都听古典音乐的人与每天听说唱音乐的人肯定大相径庭。”

卡恩又说道：“当然，说唱音乐中也有优秀的作品。”卡恩同时也是一位很有造诣的爵士音乐家。“但我想说的是，Pascal 语言塑造了最优秀一代程序员的思维方式，这正是从 Algol 发展到 Pascal 如此重要的原因。”

Algol 就像是计算机领域中的拉丁语，不仅能衍生出其他语言，而且能用于教授语言结构和语法。斯特劳斯特卢普承认，他曾想过借鉴 Algol 中的一些想法，但他太注重实际了，不愿意使用一种过于古典老旧的语言，因此选择了源于 Unix 文化中的衍生产物，即 C 语言。对于那些与计算机打交道的程序员来说，C 语言是一种实用的编程工具。但是，在贝尔实验室做复杂的模拟测试时，他也想过用其他的方法。他借鉴了 Simula 语言中的一些元素。Simula 是一种 20 世纪 60 年代起源于挪威的编程语言。开发 Simula 是为了研究特定网络中的流量，如火车票售票亭的人流，生产线上的产品，进出港的轮船数量，计算机程序中的数据流量等[12]。Simula 是由两位科学家克利斯登·奈加特和奥利–约翰·达尔于 1962 年至 1967 年在位于奥斯陆的挪威计算机中心共同完成的。

Simula 产生于“运筹学”，利用模拟等统计学方法分析和解决组织性问题。从理论上看，运筹学似乎是一种中性的技术，通过平衡数据流量的方式来提高

日常工作的效率。而实际上，它是社会工程学与管理科学相结合的产物。人类行为经过研究，就可以被破解并加以控制。因此，我们不难理解，为何苏联会在一段时间里十分热衷于将 Simula 应用在政府的乌拉尔大型计算机上。

在挪威，奈加特渐渐注意到，借助于他的发明来完成工作损害了劳动者的利益。他指出，其结果是“机械的重复性劳动越来越多，劳动者无需掌握太多的知识和技能，工作没有弹性，工作压力越来越大”。奈加特对挪威工会抱以政治同情心，因而陷入了“道德困境”。他说：“我意识到我开发的新技术严重影响了其他人的生活，尤其是那些与我站在同一政治战线上的同胞们。”于是，奈加特与势力强大的挪威工会携手，帮助他们与公司和政府签署“数据与技术协议”，要求在引进和使用计算机技术方面给予工会一定的发言权。

斯特劳斯特卢普感兴趣的是 Simula 的技术，而非其政治立场。他在 Simula 中看到的“类”是不同类型的数据，因此 C++语言起初称为“带类的 C”。除了“类”，Simula 还有两种相互关联的概念，即继承和对象，这使程序员能够运用明确定义的模块来编写软件。编程语言与英语、法语等“自然”语言一样，都是表达观点的媒介。斯特劳斯特卢普就认为 Simula 是一种能够清晰表达复杂编程思想的优雅语言。

Simula 的这些概念帮助程序员对不同种类的数据进行标注，再将数据按照逻辑层级组织到一起，这样信息就能够从一个数据类流动到另一个相关联的数据类。例如，程序员建立了一个叫做“员工”的数据类，其中的每一条数据称为“对象”，所以，出于编程的目的，一位叫玛丽·史密斯的员工就是一个对象。接下来，程序员可以设定每一个对象的属性（如姓名、年龄、社会保险号码、工资等）。同时还有一些称为“方法”的模块，可以对每一个对象进行操作（如雇用、解聘、加薪或者变更医疗福利计划等）。数据类下面还可以有子类，例如在“员工”数据类下建立子类“经理”。子类中的对象可以继承父类的特征。因此程序员只需说明“经理”所具有的特殊性，这样员工的一般特征将被继承下来。

这种以逻辑层级的形式编排数据对象的编程方法称为面向对象编程，而Simula 就是首个面向对象的编程语言。虽然 Simula 本身只是一种小众语言，但由于它将面向对象技术带到了编程中，所以产生了深远的影响。后来，施乐公司帕洛阿尔托研究中心的阿兰·凯伊和他的同事将此理念进一步应用到Smalltalk 之中。Smalltalk 也是一种面向对象语言，它将点击图标计算技术引入工作站中。此后风靡一时的 Java 也是一种面向对象的互联网编程语言。

斯特劳斯特卢普的 C++语言在最大限度上将 Simula 的理念植入编程领域。20 世纪 70 年代初，他在丹麦首次接触到了 Simula，而他对其真正产生兴趣则始于剑桥，他在那里研究因跨网络计算产生的软件问题。就在为缺少必要的研究工具而感到困扰之时，斯特劳斯特卢普意识到 Simula 可能正是他所需要的。“你必须向后模拟，而后才能做分布式计算的研究，”他回忆说，“你不可能出去买十几台计算机，然后把它们连接起来使用，那样成本太高了。”

从 20 世纪 40 年代末开始，剑桥大学的计算机实验室就凭借其杰出成就而成为了欧洲计算机研究的支柱。1949 年，剑桥设计并制造了世界上第一台存储程序计算机；几年后，在软件“子程序”的研究工作中又建立了一种模型，能够高效地重复利用代码块。剑桥大学计算机实验室是由莫里斯·威尔克斯建立的，直到 1980 年，他一直都是该实验室的主任。当时就是他和他的继任者罗杰·尼达姆主持了对这位丹麦年轻人的入学面试。直到现在，斯特劳斯特卢普还记得那次的面试既漫长又令人兴奋，两位顶尖的计算机科学家轮番提问有关编程语言、操作系统和计算机设计等方面的问题。对他而言，这就像是和两个人轮番较量，“过程极其艰难，因为一个人在向你提问的同时，另一个人已经在准备下一个问题了”。斯特劳斯特卢普出色的表现使他在 1975 年进入剑桥大学攻读硕士课程，并于 1979 年取得了该校的博士学位。

据斯特劳斯特卢普回忆，20 世纪 70 年代末期的剑桥大学就如同斯坦福大

学和麻省理工学院一样，教室内外都充满着计算机研究的学术气息。斯特劳斯特卢普接触 Simula 也是出于机缘巧合。一天，他正在研究生活动中心附近的河边酒吧“校友之家”喝啤酒，无意中听到有个学生抱怨没能使 Simula 成功运行。对斯特劳斯特卢普来说，学校里竟然有地方能够运行 Simula，这可真是个新闻。Simula 的运行设备非常昂贵，大约需要 2 万美元，涉及语言本身、源代码和编译器（为了使软件在特定的机器上运行而设计的）。事实上，Simula 没有得到广泛普及的一个原因就是挪威计算机中心的财务意识，他们把自己的软件研究成果当成需要付费购买的商品，而不是可以共享的知识。这与 AT&T 公司出于自身原因而对 Unix 采取的最初做法大相径庭。例如，1973 年，高德纳有意在斯坦福大学开展 Simula 的实验，但是挪威人拒绝给予学校较大的价格优惠，所以斯坦福大学最终放弃了购买 Simula 使用许可的打算[13]。

酒吧里这位沮丧的 Simula 用户确实是在学校的某个地方从事研究工作。虽然他与斯特劳斯特卢普素不相识，却并不吝惜自己的软件。斯特劳斯特卢普回忆道：“他说‘如果你能让它运行，我就把它送给你’。”很快，斯特劳斯特卢普就成功了。有了 Simula 的帮助，他的博士研究取得了飞快的进展。有了对象和类的逻辑结构，斯特劳斯特卢普发现，Simula 对于建立复杂的计算机网络模拟程序大有帮助。他解释说，在 Simula 中，一个大型程序“好像不再是一个单独的整体，而是被分成了若干个小程序，因此编写、解读和错误排查等都更加容易”。

但是，顺畅的研究工作并没有持续太久。他的 Simula 程序是在剑桥的 IBM 360 大型机上运行的，这台机器为包括行政部门在内的全校所有部门服务。虽然 Simula 具有很多优良的特征，但在这台机器上表现得不尽如人意，原因是 IBM 360 的数据处理能力和内存有限。“它简直像头猪，”斯特劳斯特卢普笑着摇摇头说，“IBM 360 被拖累得完全无法工作了。”

行政管理部门可不能容忍学校的计算机因为一个学生的研究项目而无法正常工作，因此明令禁止斯特劳斯特卢普再次出现类似情况。他发现问题的根

源在于 Simula 的一些基本特性，例如在处理内存和误差校验时效率低下。这个难题已经不能仅仅靠对软件的修补和微调来解决了，因此，斯特劳斯特卢普被迫放弃了 Simula，重新在另一台计算机上用其他编程语言开始了他的模拟研究。

斯特劳斯特卢普选择了使用 BCPL 语言重写他的模拟程序。BCPL 是 C 语言的鼻祖，由马丁·理查兹设计。当时，马丁·理查兹在麻省理工学院为 MAC 项目工作，项目结束后，他回到了英格兰老家，在剑桥大学教书。因此，BCPL 语言经常被应用在剑桥大学的研究项目上。斯特劳斯特卢普重做了自己的程序，以便能够在剑桥计算机实验室的实验计算机上运行，使用这台机器的人并不多。与 IBM 的大型机不同，没有公司为这台机器提供日常维护、纠错和技术支持。“但这样也有它的好处，”他说，“让非专业人士望而却步了。”斯特劳斯特卢普发现使用过于精简的 BCPL 进行模拟程序的编码和调试工作也是举步维艰。

但是，斯特劳斯特卢普别无选择，因为模拟程序是他博士阶段的研究课题。他的导师对他印象深刻，不仅仅是因为他的聪明才智，更是因为他的精力充沛和不懈努力。罗杰·尼达姆还记得斯特劳斯特卢普曾经每周工作 100 小时，就是为了能让自己编写的程序在实验计算机上运行。“他是个意志坚决的人，什么都阻止不了他。”[14]尼达姆说。这段经历让斯特劳斯特卢普终生难忘，而 C++语言之所以能够问世，部分原因也在于他在剑桥学习的那段经历。“在快要离开剑桥时，”斯特劳斯特卢普在《C++语言的设计与发展》一书中写道，“我发誓，我再不会像设计和实现模拟器时那样，用不合适的编程语言来解决问题了。”

于 1979 年离开剑桥大学之后，斯特劳斯特卢普本打算回到自己的祖国，但那时丹麦没有适合他的计算机方面的工作。如今情况已大不相同，他说。那

时，对于一个计算机研究人员来说，很难想象还有比贝尔实验室更好的工作机会。他回忆说："贝尔实验室的工作模式是提供最先进的设备和优秀的工作团队，一年以后，你要拿出工作成果。"

现在看来，由于之前的经验和对实用性的追求，斯特劳斯特卢普充分利用了在贝尔实验室工作的大好时机。工具生成项目开始后不久，他就决定将Simula的对象和类及其他可继承的特征与C语言的高效结合在一起。他的想法是，能用C语言的地方就能使用C++语言。C++语言可以被放置在任何C语言的软件项目中而无需重新编码。多年来，C++语言的主要编译器是斯特劳斯特卢普设计的Cfront，它能够把C++语言的编码翻译成C语言的编码。这也就是说，如果一台计算机能运行C语言，那么一定也能运行C++语言，这为斯特劳斯特卢普的C++语言提供了极大的市场潜力。Cfront编译的C语言编码即使对C语言编程专家来说也有些晦涩难懂，但是它快速可靠，充分体现了斯特劳斯特卢普在服务现实方面的技术才能。

斯特劳斯特卢普的目标是让现有的技术更上一层楼，甚至连C++语言的名字都体现了变革的意图。在C语言里，"++"代表"增加"或加1。根据语境，也可解释为"下一个"或"继任者"。斯特劳斯特卢普采纳了贝尔实验室的同事里克·马西提的建议，使用了C++这个名字。这同时也是一个精明的市场决策，暗示C++是C语言的自然进化，而非颠覆式的革命。"这真是灵感闪现啊！"[15]前贝尔实验室主任道格拉斯·麦克罗伊评价说。

为了能更好地与C语言兼容，斯特劳斯特卢普在C++语言中做了很多让步。他故意保留了C语言中一些"危险"或者"麻烦"的特性。因此，C++语言还保留有C语言的指针机制，使程序员能够随意进入内存。这种工具虽然很有效，却也经常带来麻烦。C++语言中没有加入"碎片收集"功能，无法自动查找并删除计算机内存中不再使用的数据。一些批评家认为，这是一项重大疏忽，因为该功能是编程语言（如Java）的基本特性。

斯特劳斯特卢普解释说，C++语言支持碎片收集功能，并可以广泛使用。

首先，他对这一决定做出了自己的哲学解释："支持碎片收集功能和必须具有碎片收集功能在本质上是不同的。"其次，从实用性的角度来看，C++最初是为大多数低水平的、与机械相关的工作而设计的，如燃料喷射器的控制。在做这类工作时，程序员通常都会避开碎片收集功能，否则会导致自动化软件在日常工作中出现暂停的现象。鉴于这种工作的特定性，专业的程序员更愿意手动解码。斯特劳斯特卢普说，如果 C++语言具备了碎片收集这一标准特性，早就胎死腹中了。

虽然斯特劳斯特卢普在 C++语言中做出了一些让步，但他心中有一个明确的目标，那就是使结构化语言的设计工具与面向对象的编程成为主流。他吸收了 Algol 和 Simula 的精华，做出些许调整，再将它们放置到一种实用的编程语言当中。由于 C++语言是基于 C 语言发展而来的，对程序员来说更易于上手。斯特劳斯特卢普的聪明才智使得 C++语言成为一种能够在所有机器上快速、高效运行的编程语言。总体而言，C++语言是以往编程语言之精华和工程学上的真知灼见的完美结合[16]，布莱恩・科尼汉这样评价道 。科尼汉是普林斯顿大学的教授，也是斯特劳斯特卢普在贝尔实验室的同事，他说："当然，我们承认 C++语言还有些小的瑕疵。但是如果你一心追求完美，也就没人使用它了。假如没有本贾尼的成就，那我们今天所说的一切也只是一纸空谈罢了。"

斯特劳斯特卢普是 AT&T 实验室的董事，也是大规模编程研究小组的负责人。他多次谢绝其他公司提供的职位和重金邀请，尤其是像华尔街投资银行这样急需强大的计算机处理能力同时又面临诸多大规模编程挑战的公司。斯特劳斯特卢普对这种有利可图的工作向来不感兴趣。"我喜欢研究那些能产生深远影响的项目，"斯特劳斯特卢普说，"如果我去某个公司赚了他们所谓的'大钱'，我的初衷也只是让系统尽快按时完成工作。"

2001 年年初，一项实用性研究吸引了斯特劳斯特卢普。该研究的目的是提

高软件的可靠性和适应性。如果成功，此项研究将缩短电话网络转换时数据通信的延迟时间，或者解决用户在移动时手机信号忽强忽弱的问题。这项软件研究用到了一辆摩托艇模型，上面装有 3 台小型便携式计算机、2 个引擎、若干个太阳能板、1 个无线接收器、1 个局域网、1 个小型高压水枪和 1 个气象站。摩托艇很重，需要两个人才能将它搬运到距离斯特劳斯特卢普办公室 100 码左右的池塘里。斯特劳斯特卢普则在办公室里用笔记本电脑控制摩托艇上的每一样设备。从理论上讲，艇上的每个装置都会立即对他的指令做出反应。

在黑板上摩托艇软件算法说明的旁边，斯特劳斯特卢普写了几个大字："该死，这可是个软件项目！"他解释说，研究团队里的一些成员好像太注重摩托艇的工程设计了，如选配件、上漆、编写发动机说明书等。他说，Boat（摩托艇）是 Basic Object Architecture Testbed （基础对象结构测试载体）的缩写。"摩托艇只是个载体，用来练习编码、做些有用的事并搞清楚故障的原因。"

第 7 章

属于自己的计算机：PC 产业的起步及 Word 的故事

俄制的 Ural II 计算机是个笨重的大家伙，要占用一个很大的房间，大部分的编程都需要对机械控制台进行手工设置，这个控制台就像是上世纪初的收银机。在这台计算机的玻璃门和操纵台后面，橘色的亮光不停地闪烁，就像是一片海洋，其中的每一束亮光都代表着一条回路。它的电子生命为人类而跳动着，像是比特的一曲点彩画交响乐。此时，15 岁的查尔斯·西蒙尼得到允许，设计并运行一些简单的测试程序。他回忆道："我高兴得都要昏过去了。"[1]

一个周六，西蒙尼得到了一个可以在布达佩斯的中央统计局为 Ural II 编写程序的机会。他试图为"幻方"编写程序，这是一个算术谜题，其中每行、每

列的数字之和都相等。直到现在，西蒙尼还记得他为这个谜题程序设置了 50×50 个格子，在当时，这可是一个野心勃勃的计划。事实也证明，这的确是一项令人望而生畏的任务，这个程序中包含了数千条指令和数不清的漏洞。最终，西蒙尼还是成功地运行了该程序，正确地填上了所有的数字；他也慢慢将计算视为一件令人着迷、引人注目的谜题。他爱上了征服计算机带来的掌控感。

对于这位出生在共产主义匈牙利的少年计算机高手来说，Ural II 为他提供了亲密接触计算机的体验。诚然，在 1964 年，按照西方的标准，这款机器还比较初级，相当于美国 20 世纪 50 年代早期的计算机。但是，这台苏联计算机是由西蒙尼直接操作的，其方式并不像当时美国常见的程序员远离计算机：要么是分时的，你得坐在终端前与别人分享计算时间；要么是批处理的，你得把穿孔卡片交给计算机中心的操作员，由他独自操作。“在我看来，Ural 更像是一台个人计算机，”他解释说，“在我使用这台机器的时候，我是唯一的用户。我和它非常密切，机器的每一个字节都只属于我自己。”

实际上，西蒙尼早期的经历，包括近距离接触和手动设置转换，强烈地反映出两代计算机运算之间的巨大差别——20 世纪 50 年代的前 FORTRAN 语言编程和 20 世纪 70 年代的微型计算机编程的第一次浪潮。在两段时期中，编程的需求都集中在少数人身上——20 世纪 50 年代的计算机中心专家和 20 世纪 70 年代的铁杆发烧友。微型计算机或者说“个人计算机”时代的到来，对计算机运算来说是一个崭新的开始，是计算机领域的再发明。它带来了硬件方面意外的突破性进展，即可编程芯片，或称微型处理器。它是由英特尔公司的特德·霍夫为日本台式计算机制造商设计的。但那些使用希斯套件来自制收音机的电子发烧友不久便意识到，这种微处理器可以当做小型程序储存式计算机的引擎。

最初，这一创新带来的结果便是 20 世纪 50 年代房间大小的计算机被 70 年代中期面包盒大小的微型计算机的所取代。早期的微型计算机使用方便，和 50 年代的大怪物计算机一样，使用了相同的编程。对那些早期电子发烧友来说，

研发微型计算机的过程是痛苦的，但同样也是令人振奋的。这群电子迷是一个关系密切、互帮互助的小团体，他们在小机器上试运行一些程序，主要是一些简单的游戏程序。这项工作并没有什么赚头，但是，随着芯片功能的日趋强大，这些廉价的机器显然不只是些玩具了。它们能做真正的计算机才可以做的工作，这也就潜在地让普通人拥有了掌握计算机运算的能力。随后的个人计算机革命在很大程度上都应该归功于这种企业家精神和计算机科学的合二为一。个人计算机产业是由外行人开启的，那些电子发烧友及年轻的企业家新贵，如苹果计算机公司的史蒂夫·乔布斯及微软公司的比尔·盖茨都算是外行人。但是，为了将计算机运算普及给大众，他们还需要借助于学校及研究所的计算机科学成果。最初选择的编程语言是达特茅斯大学开发的 BASIC，后又增加了美国电话电报公司贝尔实验室开发的 C 语言和 C++语言。用鼠标的指向与点击来移动与控制计算机桌面上的图标，这种人们与现代个人计算机互动方式的设计可追溯至施乐公司的帕洛阿尔托研究中心及斯坦福研究所所做的工作。

查尔斯·西蒙尼的生活与事业为我们描绘出了一道漂亮的弧线，它贯穿了从科学技术的黑暗时代到个人计算机时代，从计算机的科学研究到企业家成果的整个计算机运算时期及文化。西蒙尼在 17 岁时就离开了匈牙利，向西方逃亡，先去了丹麦而后来到加利福尼亚；在这里，他先后就读于加州大学伯克利分校和斯坦福大学。20 世纪 70 年代，他供职于施乐公司的帕洛阿尔托中心，开发出了 Bravo 程序，这是一种全新的书写和文本编辑程序。1981 年，他加入了微软公司，引领这家公司在一些领先产品上的科技进步，其中包括基于 Bravo 程序的 Word 文档编辑器，而 Word 如今已是世界上运用最广泛的程序之一。当然，其他人也对研究做出了巨大的贡献，但是，西蒙尼所在的施乐公司帕洛阿尔托研究中心团队在接下来的 25 年里引领着个人计算机领域重大的变革。单单西蒙尼在 Bravo 程序设计上的成果便是对这一领域最重要的贡献，它决定了上百万人如何在计算机上创建文档，包括商业备忘录、小说等。西蒙尼并不是一个企业家；他没有创办过任何一家公司，也不是公司高管，甚至名字也不

被人熟知。但是，他的确帮助盖茨创建了微软公司，尤其是在该公司创始之初的那些年。他是微软公司主要产品的设计者，公司技术智囊团的重要成员，编程天才和研究人员的招募者。西蒙尼是一个特殊的程序员，他将科技和软件经济学完美地融合在了一起，并使之成为一种时尚。当年他离开共产主义匈牙利时身无分文，而如今已是身价数十亿的富豪。

早在孩童时代，西蒙尼看待事物的方式就与众不同，会冷静地分析周围的事物。他是拼装玩具的狂热爱好者，但是在当时的匈牙利东区，即便是零部件都非常罕见，更别说一整套玩具了。尽管当时这种拼装玩具十分稀缺，西蒙尼还是成功地拼凑出了一辆拥有四速变速箱的“小轿车”。“我的轿车看上去并不像轿车，但是它能像轿车一样工作，”西蒙尼回忆说，“我只是在利用拼装轿车进行抽象思维。”一旦拼装完成，他便对已完成的机械发明失去了兴趣。真正的吸引力在于解决问题的过程，拼装玩具的各个部件都只是达成目标的媒介。“我永远不会围着一辆小车发出‘Z-Z-Z-Z-Z’的声音，”西蒙尼边说边在地板上空移动他的手，好像他正拿着一辆比赛中的玩具车一样，“我很奇怪吧。”

西蒙尼 14 岁时，《布达佩斯》时报上的一篇专栏文章给他留下了深刻的印象。那是一位名叫安东尼奥·葛兰西的意大利共产主义者写的文章，被翻译成了匈牙利语。根据西蒙尼的回忆，葛兰西的这篇文章讲述的是伟大的共产主义领导人列宁和木柴的故事。一位生活在俄国乡村的老太太家中的木柴就快用完了，此时正值寒冬时节，供给很有限，她的木柴撑不过整个冬天。绝望中，她写了一封信给列宁同志，诉说自己面临的困境。列宁读了老人的信后亲自处理了这件事，于是老人有了足够过冬的木柴。这是一个讲述共产主义善行的故事，但是在当时年轻的西蒙尼看来，这是十分可怕的。他说：“我读了那篇文章之后想：‘一个人会不会被冻死，难道是由领导人看不看你的信决定的吗？’我要尽快逃离这种体制。”

具有讽刺意味的是，他的逃离正是借助于这个共产主义政府的数据统计机构，因为正是在那里他接触到了计算机和计算机编程。青少年时，西蒙尼对与机械有关的事情就很感兴趣，他常参加布达佩斯的商品展，参观那些机器能够“给我新想法”，他回忆道。大多数时候，在这些展览会上展出的只是挤奶机、拖拉机或者手提钻之类的东西。但是，当听说并初步了解了计算机后，他就想要得到一台。当时，只有政府机构才有计算机。西蒙尼的父亲卡洛里是一名物理学家、大学教授，他教过的学生中有一位是中央统计局的首席工程师。这位名叫佐尔坦·容博克的工程师让这个少年进入了统计局，并耐心地向他讲解 Ural 和编码程序的基础。于是，20 多岁的天才工程师容博克就成了西蒙尼的启蒙导师。容博克在 30 多岁的时候死于哮喘导致的呼吸障碍，但是他对西蒙尼影响深远。西蒙尼说：“将所有的一切看做一种计算过程，这种理念源于容博克。”

西蒙尼不屑自己在统计局所做的编程工作，在科学社会主义中央计划经济下，这些编程用于计算原材料和产品的装运。他说：“有市场价格，这些问题在一毫秒内就能解决。”但是，对于这个聪明、有很强求知欲的年轻程序员来说，中央统计局的确是一个极好的实验室。他发现，Ural 是一台抽象计算机，因为在 Ural 上编程是一件令人疲惫和泄气的事情，但同时 Ural 也确实是个了不起的多功能机器。“在某种程度上，我找到了自己的终极拼装玩具，”西蒙尼指出，“有无穷的拼装组合。”

西蒙尼开发的首个专业软件是 Colur（Code Language for Ural，适用于 Ural 的简单编程语言）。它有 5000 行，编程采用原始的八进制系统，但是，它的确能让程序员用编码来书写运算过程，编码只能被程序员理解，而不能被计算机计算，然后这种运算被翻译成一种可以被 Ural 执行或运算的形式。正是由于其包含编译程序，Colur 属于编程语言的一种。它的价值在于为匈牙利政府的编程人员节约了时间。由于使用的是八进制编程，Colur 并不完美，但依然是成功的。“它是我的 FORTRAN。”西蒙尼说这句话的时候带着一丝骄傲。西蒙尼在西雅图郊区湖边小屋的保险箱里存放的不是现金、珠宝，也不是债券、股票，

而是两样东西：表明他美国公民身份的国籍文件以及他编写第一个应用程序Colur的纸带。

对于西蒙尼嘲笑共产主义集体主义约束这件事，后来认识他的那些人一点都不感到吃惊。西蒙尼是一个高度个人主义者，不能严格地划分到任何一类人当中。举例来说，在施乐公司帕洛阿尔托研究中心工作时，西蒙尼便以他有趣的过去、广泛的兴趣以及嗜好西方的某些光鲜玩具而被人熟知。他开着一辆捷豹 XKE 跑车。“查尔斯从来都不是一个戴着厚眼镜片的书呆子，”[2]西蒙尼在施乐公司的同事、硅谷风险资本家约翰·肖奇回忆说，“他还有很多不同的行头。”从外表看，一双深邃的眼睛，大男孩般的面庞，高挺的鼻梁，人们常说西蒙尼很有拿破仑的风范，尽管他比拿破仑要高大、匀称得多。

不管是在家宴客，出门旅游，还是交朋友，西蒙尼都不是一个热情的参与者，很多时候更像是一个独行侠。西蒙尼终身未娶，但是他与杂志企业家玛莎·斯图尔特是多年的朋友。当她的公司于 1999 年在证券市场上市时，西蒙尼获得了最大份额的“朋友及家人”股份[3]。西蒙尼的房子是高度现代主义的杰作，房间的内部给人视觉上的冲击。尽管如此，大多数人还是发现，住在这样的房子里即使不打冷颤，也会有些吓人。屋内所有的物体表面都是干净圆滑的；没有任何的植物、照片、杂志或随意摆放的物品。“这是一个实实在在的问题，没有人送我礼物，而我在买东西时也十分谨慎。”西蒙尼说话的时候微微笑了一下，暗示他认为美学上的纯净大于一切。他的家就是他自己的城堡，他并没有仔细想过要去容纳其他人。他的朋友兼同事巴特勒·兰普森说道：“只有查尔斯才会建那种占地两万平方英尺却只有一间卧室的房子。”[4]

西蒙尼逃离匈牙利始于一个很小的叛逆行为：他把头发留得很长，结果被

学校赶回了家。当时他刚上高三。那天傍晚，他父亲卡洛里・西蒙尼回到家，问他想要什么生日礼物。西蒙尼回答说“退学”。父亲并没有责令他听话、剪短头发并返回学校，而是平静地向他解释了当时匈牙利国内有两种可以不在教室学习的情况：第一种就是学生患了很严重的学习障碍，另外一种是这个学生被认为是某些方面的天才。

早在那时，西蒙尼在中央统计局的工作已充分展现出他的计算机天赋。因此，在父亲的帮助下，他向教育部门递交了申请，声明自己可以在家中自学，而且每年进行两次例行考试，这样他就可以提前一年毕业。申请递交到政府部门之后，这种特殊的安排让西蒙尼不受约束地去学习计算机知识。他在家中自学，跳级完成功课，逮着工夫就学习英语。年满 18 周岁就得去服兵役，他只有一年的时间了，因此他制定了一个计划。在布达佩斯的一个商品展销会上，他接触了一些丹麦计算机公司的工程师，并且向他们询问了一些关于计算机的问题。他利用这些信息制作了一份他所谓的“活简历”，这是一个带有编程样本的纸带盘，并打算在下一场商品展销会时将它交给这家丹麦公司的代表。西蒙尼回忆道：“我给了他这个纸带盘，只简单地对他说‘把这个转交给你老板。’”

几个月后，这家丹麦公司的总经理尼尔・伊瓦尔・贝克到布达佩斯出差，并邀请这位少年共进午餐。西蒙尼自信地前来赴宴，在此之前，他已研究过丹麦 Gier 计算机及其编程程序。他们聊得非常愉快，交换彼此的信息，机器和程序看起来也不那么复杂了。西蒙尼说：“在那个相对简单的年代，一个人想知道计算机和软件的所有信息是可能的。”西蒙尼给总经理留下了深刻印象，所以不久之后，西蒙尼便收到了一封邀请其前往丹麦工作一年的邀请函，还有一张去哥本哈根的机票。匈牙利当局允许西蒙尼离开，但条件是他需要参加大学入学考试并且被录取。在当时的共产主义匈牙利，大学教育是一种对特定群体的嘉奖，这个群体被认为已选定了终身职业。很显然，一个大学学习机会虚位以待，再加上全部家庭成员都在匈牙利，当局料定这个少年会在规定的时间返

回布达佩斯，先服完兵役，然后在匈牙利完成大学学业。

当然，西蒙尼并没有这样的打算。只有他的父母知道他想去西方国家，连弟弟塔马斯都不知道。一年后，在应该入伍的当天，西蒙尼并没有出现，于是，军队官员来到西蒙尼的家中，询问他的下落。正如西蒙尼讲述的那样，他的母亲苏珊娜答道："我想，他正在哥本哈根和旧金山之间的某个地方。"他的离开给家庭带来了一些影响，为此，他的父亲丢掉了布达佩斯科技大学的教授工作。不过，他父亲后来编著了一部经典的物理学文化史，该书经过多次印刷，还被翻译成了德语，书名为 *Kulturgeschichte der Physik*。"对于父亲来说，丢掉工作并不是一件好事。但正因为如此，他写出了平生最好的著作，"西蒙尼说，"我的父亲活出了他自己，同样他认为我也应该有自己的生活。对于这一点，他十分清楚。他从不让我对自己所做的事情感到后悔。他永远也做不了我所做的事情，但是他帮助我做到了那些事情。"

西蒙尼在 1966 年 7 月 17 日到达丹麦，几乎身无分文。他的丹麦老板给了他 500 克朗（在当时大概相当于 100 美元），并带他共进午餐和晚餐。第二天，西蒙尼便开始在 A/S Regnecentralen 公司工作，这家公司是政府参股拥有的，旨在促进丹麦计算机技术的发展。西蒙尼是所在编程团队的第三位成员；起初，团队成员都以质疑的眼神来看待这个年轻的小子。他在上班的第一天就开始编写代码。"这是一个破冰的举动，"西蒙尼指出，"这让他们认定我是同类人。"

在丹麦，西蒙尼接触到了关于 Algol 的结构型编程原则。虽说 Algol 由于在大多数计算机上运行缓慢而臭名昭著，但是在丹麦的 Gier 计算机上它可以高效运行，这一点多亏了 Regnecentralen 公司编写的编译程序。这个编译程序是由彼得·诺尔编写的，他是丹麦著名的计算机专家，也正是他帮助西蒙尼移民到了美国。诺尔所写的个人推荐信实实在在地帮助了西蒙尼：这个没有高中毕

业成绩单，也并没有参加大学理事会考试的匈牙利小子被加州大学伯克利分校录取。

在伯克利，西蒙尼的专业是数学工程而不是计算机编程。他说："我不需要再学习计算机了，我已经是专家了。"如果他需要一个"A"来提高自己的学业平均分，那么他将会利用伯克利的一项计划，那就是伯克利的学生可以"挑战一门课程"：不需要听课，只参加期末考试。如果是这样的话，他将会参加计算机科学这门课的考试，他一定会微笑地走出考场的。

西蒙尼很自然地被伯克利的计算机中心吸引了。在计算机中心工作可以获得每小时 2.95 美元的报酬，但是所要做的工作并不仅仅是例行公事。西蒙尼的工作证明了他想尝试更高级的计算机运算。一天，年轻的伯克利教员巴特勒·兰普森要求西蒙尼编写一个编译程序，将原本用 Snobol 语言编写的程序编译成机器码。Snobol 是 20 世纪 60 年代早期贝尔实验室开发的一种符号列表处理语言。那时候，西蒙尼在设计和编程方面已经有了相当多的经验，这些经验技巧都是他在学习 Ural 时积攒下来的，而这正是设计一个快速、敏捷编译程序所必需的；这些经验也在他所设计的编译程序中得到了充分的体现。后来成为计算机研究领域领头人的兰普森对这些编码程序以及这位匈牙利程序员印象深刻。在接下来的十年里，他们并肩工作，包括在施乐公司帕洛阿尔托研究中心共同开发 Bravo。

兰普森是伯克利大学 Genie 项目组的技术带头人之一，该项目研究由国防部先进研究项目署支持的分时系统。Genie 项目由政府出资支持，目标是要构建一个小规模的分时计算机系统，以证明这个系统在技术上和经济上都是可行的。它是麻省理工学院庞大的分时项目 Multics 的微型版本。运用具有大型中央处理机的计算机，Multics 项目旨在将 300 个用户同时接入计算机。Genie 项目团队使用了一个较小的科学数据系统 930 计算机，并且将它的软硬件进一步

升级，这个分时系统一次最多可以同时连入 20 个用户。Multics 的创建是计算机科技上的一次胜利。Genie 项目团队在胜利中依然保持着清醒，他们决定在分时技术上更进一步。1968 年，他们创办了一家名叫伯克利计算机的私人公司，雄心勃勃地要开发出可同时接入 500 个用户的计算机系统。此后他们在华尔街筹资，并开始招兵买马。

1967 年 11 月，西蒙尼来到伯克利，没能赶上 Genie 项目。但当时，伯克利计算机公司还处于起步阶段，他的导师巴特勒・兰普森不久后便邀请他加入伯克利公司。伯克利公司的计算机被证明是诠释弗雷德・布鲁克斯所谓“第二系统”综合征的经典范例。Genie 项目在设计上只是一种扩充，而 BCC 500 项目仍只是一个美好的愿望。伯克利计算机公司一下子想要做的事情太多，于是资金很快枯竭；到 1970 年年末，华尔街的投资家也失去了耐心。但是，对于西蒙尼来说，伯克利计算机公司不仅给了他与兰普森共事的机会，也让他认识了查尔斯・萨克尔，后者是 Genie 项目组成员之一，也是一位计算机奇才。和兰普森一样，萨克尔是一位熟练的计算机系统设计者，但是更倾向于硬件设计。在伯克利计算机公司，西蒙尼负责编写微代码，这只是软件设计的基层，更像是电路本身，不过它是可以被编程的，编程者可借此自由调整机器。

即使在那时，西蒙尼也表现出了很高的天赋。他每晚都穿着“调试服”[5]，黑色的网状衫、半透明的黑色紧身裤，努力地清除软件中的漏洞。此外，西蒙尼也是排除漏洞的高手，正像萨克尔说的那样，“要是没有他，进展就会很慢”。

的确，直到西蒙尼于 1976 年获得斯坦福大学博士学位，他一直在做全职工作；从表面上看，他同时还是一位全日制学生。西蒙尼从未怀疑哪段经历更加重要，不管是个人方面还是专业方面。他说：“与巴特勒和查尔斯・萨克尔共事多年，其价值远远胜过好几张学位证书。”当西蒙尼于 1972 年加入施乐公司帕洛阿尔托研究中心与兰普森和萨克尔共事时，他还没有获得加州大学伯克

利分校的学士学位。就这样，没有获得学士学位，没有工作许可证，也没有绿卡或永久居住签证，西蒙尼成了施乐公司帕洛阿尔托研究中心的一员，直到 1974 年他才获得绿卡。西蒙尼说："不过，施乐公司帕洛阿尔托研究中心渴求优秀的程序员。"

---

20 世纪 70 年代初期，施乐公司帕洛阿尔托研究中心开始研究著名的"梦想计算机"，这是一款供个人工作或娱乐、具有高分辨率屏幕的小型机器，即"个人"计算机[6]。对于个人计算机时代的孩子们来说，个人计算机是一件理所应当的物品，很难向他们讲清楚这种观念在当时多么具有革命性。事实上，当时的人们不仅广泛地认为这个观点不可行，更认为它是堕落的，因为这种做法对于那些价值不菲而且珍贵的计算机资源是一种浪费。西蒙尼说，对主流思想来说，"那种一人拥有一台计算机的想法是令人憎恶的"。但是，施乐公司帕洛阿尔托研究中心那些富有远见的领导者，譬如罗伯特·泰勒和阿兰·凯伊，都是个人计算机概念的积极拥护者。同样重要的是，像巴特勒·兰普森和查尔斯·萨克尔这样的工程师也逐渐认识到将这种愿景转化为现实是可行的。半导体科技、存储及显示技术的迅速进步将个人计算机这个目标变得触手可及，至少在研究层面上是这样的。

萨克尔带领硬件设计团队设计了现代个人计算机的鼻祖 Alto。第一代 Alto 个人计算机诞生于 1973 年春天，这种计算机拥有创新型位映射显示技术，以确保屏幕上的每个图像元素（像素）都可以受存储于 Alto 中的编码程序控制。这种进步打开了一扇大门，允许使用者用不同大小、形状和风格的图表与文本来"覆盖"显示屏，进而创造出之前不可想象的视觉灵活性。新硬件计算效率的提高扩大了软件的适用范围。按照西蒙尼的说法，位映射技术给软件和硬件之间的力量平衡带来了巨大转变。"硬件并不能决定你的所见和所为，"西蒙尼说，"硬件的作用就是显示比特，它是为软件服务的。"20 世纪 70 年代早期，

大多数人与计算机的互动都要通过一个分时系统的 IBM 终端。IBM 的终端显示屏只能满足视觉上的最低要求。它用一种 IBM 认为合适的字体显示数字和全部大写的字母。而 Alto 展现的则是另一种方法，不论在技术提升方面还是在随后的成本节约方面，这种方法都是很重要的。西蒙尼说："这是一种完全不同的思想方式，是对软件和硬件的分离。"

然而，1974 年年初，Alto 计算机仍需要借助软件向大众展示它能做什么。巴特勒·兰普森认为，从创建编辑文件的程序入手将会是个不错的开始；用今天的话说，这种程序就是文档编辑器。有一天，西蒙尼闲逛到他的办公室，兰普森刚刚画出这款文本编辑程序的粗略设计图。兰普森向西蒙尼解释了他的想法，而西蒙尼听完之后立即被吸引了，不停地翻看着那几张写满运算法则和图表的黄色纸片。西蒙尼向来都对施乐公司帕洛阿尔托研究中心富有挑战性的项目很感兴趣，尤其是还能帮助他获得斯坦福大学博士学位的项目。

长期以来，西蒙尼对编程技巧很是着迷，用各种方式改善编程过程、工具，提高编程技巧。他的博士论文题目是《元编程：一种软件制作方法》。西蒙尼的元编程方法有很多省时的特点，其中就包括匈牙利命名法（Hungarian）的使用，它使用辅音符号进行速记以简化信息名称。西蒙尼曾半开玩笑地说，他的匈牙利命名法就像是乔叟时代的古英语；Hungarian 不仅是指他的祖国匈牙利，同时也是一种幽默的叫法，当首次接触这种标示时，编程人员也许会认为那些看起来很奇怪的字母组合是匈牙利语。很多编程人员学习过后都发现匈牙利命名法很有帮助。西蒙尼教微软的编程人员使用这一命名法，而且 Word 与 Excel 中的部分代码都是用匈牙利命名法完成的。最新版本的《牛津英语字典》中收录了西蒙尼的这一命名方法，将其作为 Hungarian 词条下的一种解释。

元编程这个概念的本质是将编程人员分组；他们受各自组长的指挥，而这个组长就是所谓的元程序员。这种方法包含了对成员严格而统一的分工：精英程序员为其下属出具详细的计划书，而这些下属就像技术员那样负责编写代码。元编程也是"软件工程"的一项内容：将严格的工作计划书和过程控制等

工程规范应用于技巧性编程之中。在微软，西蒙尼曾打算把元编程制度化，但是随后摒弃了这一想法。这不符合当时的公司文化，这个规模较小却发展迅速的公司试图鼓励每位员工的一举一动都要向企业家看齐。“我们的确使用了匈牙利命名法，团队结构也很紧凑，这些都是元编程的必要元素，”西蒙尼说道，“但是，我们并没有尝试以精细化的方式来组织公司，雇用那些廉价的程序员和技术员，让他们听命于元程序员。那种方式是不对的……我们并没有尝试以元编程的方式来组织公司，然后对其否定。我们压根就没有用过这种方式。”

尽管如此，回到施乐公司帕洛阿尔托研究中心后，西蒙尼依然对元编程持乐观态度，并且急切地想完成这个主题的博士论文。那天，当走进兰普森的办公室时，他就已经准备好了一个提升软件生产率的实验计划，这个计划还需要两位斯坦福大学生的帮忙。第一次的实验被命名为 Alpha。对西蒙尼来说，兰普森用几篇稿纸向他展示的文档编辑器项目可能会成为元编程的另一个试验台，而兰普森，带有一些汤姆 · 索亚的冒险精神，则鼓励了这个计划[7]。“查尔斯最初的动机并不是要编写一个文档编辑器，”兰普森回忆说，“他只是想测试自己的元编程思想而已。”就这样，作为 Alpha 的后继者，第二次元编程实验 Bravo 项目开始了，它将对软件领域产生长久的影响。而不久之后，文本编辑软件成了工作的重点，重要性远超元编程，但是他们依然保留了元编程这个名字。

Bravo 程序是由兰普森和西蒙尼共同设计的。兰普森说，这样的合作是“对高层次设计持续讨论”的结果，因此“很难说明白这个程序到底是源于谁的思想”。西蒙尼是主要负责将这一思想编码化的程序员。Bravo 是巧妙而又机敏的程序，利用计算机储存中的大部分资源，通过显示屏向用户展示它的魔力。其中一个巧妙的想法是兰普森提出的运算法则，当一个文件表示为构件表形式时，充分利用 Alto 的存储资源。构件表的概念源于施乐公司帕洛阿尔托研究中

心的程序员杰伊·摩尔。该运算法则保证文件以文本分程序的集合（构件的表）形式储存在计算机内存中，而非表示为储存在内存中的每个字母或者文字。如果一位作家想把一则短篇小说中间部分的一段话移至文章的结尾，那么将会出现 3 个构件：被移动段之前的文章部分，被移动段和文本剩余部分。对于计算机而言，简单地说，1-2-3 块的模式将变为 1-3-2：这是 3 个模块之间的互换，而不是上千文字的移动；其差别对于人类而言，就像是将物品整齐地装进几个包裹里，而不是随便抱在胸前。

随后，西蒙尼对这一程序做了进一步的改进。他认识到，在任何一个编辑文档中，大部分内容都是未改动的；把正在改动的部分从文档中独立出来，并分别用软件束包裹起来，将在很大程度上节约计算机运算资源。用户可以通过显示器看到修改后构件表的虚拟文档，但是直到编辑工作完成，这些文本才被存储到计算机里。西蒙尼说，“构件表让你在不改变计算机文件的情况下对文档进行修改”，这进一步地节约了运算资源。

那些“节省”下来的运算资源可用于传输内容丰富的文本和图表至用户的显示屏。通过节约每个字母或文字的字节占位（起初是每字节 25 比特，后来增至 100 比特），西蒙尼成功地使文字具有丰富的文本形象，包括黑体、斜体、下划线、字号、字体甚至颜色。他说：“这样能确保在以后的 30 年里有完善这款软件的空间。”随后，他又编写了对这些属性进行高效编码的基本运算法则，以便 Alto 识别与处理。向大小任意的文字区域发出“信号”，即格式操作符，格式就会改变。这些程序都被塞进了计算机内存中，与现在的标准相比，Alto 的处理能力和存储能力都是极其微小的。Alto 的微处理器运行速度小于 6 兆赫，而 2001 年标准台式计算机的运行速度在 800 兆赫以上。Bravo 在设计上包含许多工程艺术。西蒙尼说：“发现问题本身并没有什么，关键在于明白这些问题可以解决以及如何解决。”

Bravo 程序集眼界、新算法与低级语言代码于一体，充分证明了西蒙尼在计算机方面的天分。与他共同设计 Bravo 的巴特勒·兰普森评价：“从比特层

面到高级软件设计层面，查尔斯都是一位非凡的程序员。”查尔斯·萨克尔是他在施乐公司帕洛阿尔托研究中心的另一位同事，曾描述西蒙尼在 Bravo 程序设计上的成就为“从其他人身上借鉴几个不错的点子，加入一些自己的想法，就构建出了一个伟大的概念统一体”，还能在非常有限的计算机资源下高效运行。萨克尔还说：“在我漫长的职业生涯里，很少遇到像查尔斯这样的综合性人才。”

设计 Bravo 时，兰普森和西蒙尼专注于基础编程。起初，用户界面很粗陋，运用的是已被计算机界面设计师摒弃了的设计，就是让人摸不着头脑的“模式”。计算机用户要么使用文本模式，要么使用命令模式，这两种模式只要有一点混淆就会以悲剧告终。有个经典的笑话用 edit 一词来说明模式的危害。在文本模式下，这四个字母会简单地出现在屏幕上；但在命令模式下，输入字母 e 会选定整个文件，d 将其删除，i 命令计算机插入新文件——所以，倒霉的用户只能眼睁睁地看着自己的工作成果瞬间消失，取而代之的是一个孤零零的字母 t。

关于 Bravo 程序的这一问题，西蒙尼很快便找到了解决方法。在此之前，施乐公司帕洛阿尔托研究中心其他部门的两位研究员拉里·特斯勒和蒂姆·莫特花费近两年的时间开发了一个新的文本编辑用户界面，名为 Gypsy。他们在波士顿的一家教科书出版社里，针对人工编辑工作做了大量的调查[8]。现代文档编辑器程序中的方法及使用的图示词汇表都源于他们的工作，譬如以剪刀、胶罐的图标代表剪切、粘贴命令，以及定向点取的鼠标操作。

Bravo 程序在 1975 年 3 月搭载了 Gypsy 界面，变为 Bravo 3。西蒙尼说：“Bravo 3 才是真正的赢家。”1975 年夏天，一个奇怪的现象出现在施乐公司帕洛阿尔托研究中心。研究员的朋友、亲属和邻居都开始使用 Alto 来制作并打印文档，比如家长教师协会的通知、个人信件、大学论文、商业计划书以及其他

各类文件。他们的参与在很大程度上是受到欢迎的，他们就像是在为未来办公软件做实验的小白鼠。“这也许是那些对计算机并不感兴趣的大众第一次接近并使用计算机，” 西蒙尼这样回忆道，“他们并不是计算机迷，也不是铁杆电子发烧友。他们只是想通过计算机做点事情。那时我便对自己说：‘哇，这个行业将大有可为。’”

这种吸引力源于更易操作的计算机，其文本编辑程序可以在用户界面上显示文本和图表的各种字体及风格。同时，这些文本和图表也可以依照界面的显示直接打印出来；虽然就目前来看，这种技术已经司空见惯，但在当时，这的确是一项突破性的进展。这就是人们所熟知的 WYSIWYG，发音同 wizzy wig。这是喜剧演员弗利普·威尔森说过的一句著名台词的首字母缩写：“所见即所得”（What You See Is What You Get）。

Bravo 程序带给个人计算机的 WYSIWYG 编辑方法催生了桌面出版业，也改变了专业编辑、出版商、插图艺术家的工作方式，减轻了他们的负担并提高了劳动效率。与此同时，桌面出版也让数百万业余爱好者制作出了自己的报纸、本地杂志及贺卡。和 Alto 一样，Bravo 就其本身而言并不是商业性产品。即便如此，Bravo 仍是第一代“杀手级软件应用程序”，它使普通人想要使用个人计算机；在那时，“杀手级应用程序”（killer app）这个词语还远远没有出现。

早在 1976 年，康涅狄格州斯坦福德市的施乐公司总部就已认识到 Alto 是一项有潜力的技术。施乐公司的计划是要让 Alto“步入正轨”；查尔斯·西蒙尼认为这又是一个“第二系统”综合征的例子，他的同事查尔斯·萨克尔简洁地将其称作“求大主义”。1978 年，施乐公司组建了一个小组，负责把 Alto 卖给一些大客户，让他们率先尝试一个更好的计算机办公世界。此时距离 Star 走向市场还有一段时间；Star 是施乐公司继 Alto 之后推出的信心之作。西蒙尼很高兴地加入了这个新业务单元。约 1500 台 Alto 计算机被制造出来，并有选

择性地送到了诸如波音这样的大公司，甚至送到了吉米·卡特[①]的白宫。西蒙尼在Star上表现出来的直觉被证明是精准的。1981年，首次推介Star时，它的表现就令人印象深刻。在当时看来，Star的功能十分强大，具有支持位映射的大型显示屏，能够非常清晰地展示文本和图像。但是，它的售价超过了1.6万美元，而且瞄准的市场份额非常小。此时，计算机领域最具信誉的品牌IBM推出了针对商业人士研发的相对便宜的个人计算机。

20世纪70年代的大多数时间，施乐公司帕洛阿尔托研究中心的研究员基本都认为像Altair 8800和Commodore PET这样的微型计算机只能算作是计算机爱好者和孩子的游戏机，并不是真正的计算机。对于施乐公司帕洛阿尔托研究中心的研究员来说，这些计算机不属于政府、大学或者公司研究所的研究产物；那些制造微型计算机以及为这些计算机编程的人也不是科班出身的专业人员。但是，20世纪70年代末，施乐公司帕洛阿尔托研究中心的一些年轻研究员发现，微型计算机现象极具潜力，足以改变整个计算机领域。

对于西蒙尼，在硅谷的一家德国餐馆与保罗·海克尔共进晚餐之后，他清楚地意识到了这一点。海克尔是施乐公司帕洛阿尔托研究中心的一名顾问，与西蒙尼相识多年。他提议西蒙尼去他家中看看苹果二代计算机上安装的新软件程序。随后，西蒙尼去了海克尔的公寓，并且观看了海克尔演示的VisiCalc软件，这是计算机史上第一个电子制表软件。“我被震撼了，”西蒙尼回忆道，“我一下子变得清醒了。它让我意识到施乐公司帕洛阿尔托研究中心并没有掌握全部的顶级技术。VisiCalc软件是一款具有重要意义的软件，施乐公司帕洛阿尔托研究中心从未研发出这么厉害的软件。”

看了VisiCalc软件之后，西蒙尼愈发想要离开施乐公司帕洛阿尔托研究中

① Jimmy Carter，吉米·卡特，美国第39届总统。——译者注

心，甚至有了自己开公司的念头。施乐公司帕洛阿尔托研究中心的一位硬件设计工程师鲍勃·贝尔维尔用英特尔微型处理器制造了一台小 Alto，这台机器有一个位映射的显示屏和鼠标。西蒙尼拜访了贝尔维尔，并见到了存放在车库里的这台计算机。西蒙尼满腔热情地建议他们应该合伙成立一家公司，贝尔维尔负责硬件方面的设计，自己负责软件方面的设计。贝尔维尔有点犹豫，西蒙尼也就没有过多地坚持。事实上，直到现在，西蒙尼承认自己确实没有太多的兴趣来创办自己的公司。之前他一直告诉他的朋友们他想在美国取得成功，但是并不是以商人的身份。从一开始，他就非常有信心，自己的技能足以使他变得富有。查克·萨克尔记得有一天，西蒙尼开着那辆捷豹 XKE 跑车，带着一堆价值不菲的摄影器材到施乐公司帕洛阿尔托研究中心，这些摄影器材都是他用信用卡购买的。“我提醒他偿付利息在美国是个‘阴险的陷阱’。”萨克尔说。可西蒙尼回答：“不用担心，这不是问题。我会变得很富有，还这点小钱还是很容易的。”

西蒙尼在施乐公司帕洛阿尔托研究中心无法变得富有；但是致富的机会在微型计算机领域到处都是，他决定去抓住它们。西蒙尼请教了一位离开施乐公司帕洛阿尔托研究中心并获得成功的研究员罗伯特·梅特卡夫。梅特卡夫开创了将计算机连入网络的以太网标准，并且创办了 3Com 公司，专门生产联网设备。西蒙尼觉得梅特卡夫是他认识的人当中最熟谙经商之道的人。在斯坦福购物中心的露天餐厅共进午餐时，梅特卡夫给了西蒙尼一张纸片，上面写着三四个人的名字，以及他们所掌管公司的名字。西蒙尼至今还记得其中三个人的名字：丹·费尔斯塔拉，Personal Software 公司董事长、VisiCalc 的发行商（西蒙尼认为他是“当时的顶级人物”）；格雷·基道尔，拥有微型计算机操作系统 CP/M 的 Digital Research 公司首脑；还有一个就是微软公司的比尔·盖茨。苹果公司的史蒂夫·乔布斯可能在名单之列，但那时苹果公司已经是一家相对成熟的公司，准备于 1980 年年末公开出售股份。西蒙尼想成为一家新兴公司的元老，“苹果公司并没有这种机会”。

西蒙尼首先选择了微软公司，之后再也没有去其他的地方。他在一次去波音公司的商务旅行中认识了比尔·盖茨，当时盖茨正想从施乐公司帕洛阿尔托研究中心挖几位专家。在西雅图的最后一天，西蒙尼开车来到了微软公司。微软公司坐落在贝尔维郊区的一栋银行大楼的第八层。西蒙尼以为梅特卡夫事先已电话告知比尔·盖茨自己将于两点钟拜访，但是梅特卡夫并没有提前通知盖茨。西蒙尼说："这就变成了一次彻底的自我推销，但是我并不知情。"当时盖茨正在会见一群日本客户，所以就由微软公司的二把手史蒂夫·鲍尔默来接待西蒙尼。西蒙尼随身带了一些资料，包括一家咨询公司对 Bravo 产品不吝赞誉之词的报告。虽然西蒙尼的这次拜访并没有与盖茨进行事先沟通，但是，西蒙尼的名气帮了大忙。盖茨说："我们听说过西蒙尼。"[9]

盖茨结束了与日本客户的会面，但那时西蒙尼已经快要赶不上回旧金山的飞机了。于是，盖茨亲自开车送西蒙尼去机场，他们"一路上都在讨论"：讨论有一天个人计算机将会出现在世界各地的办公室和家庭里；讨论微软公司用类似 Bravo 那样的产品进入应用软件业的计划；还讨论了微软公司的成长目标以及如何发展的问题。"我同那个家伙展开讨论，他解答了我所有的疑惑，"西蒙尼说，"这太棒了。"

西蒙尼回到加利福尼亚之后，盖茨前来拜访，他们又交谈了好几个小时。后来，西蒙尼写了一个简要的备忘录，大概有四页纸，简要总结了他们的谈话并详细地阐述了自己的想法。1980 年 11 月，西蒙尼在一台装有 Bravo 程序的 Alto 计算机上写下了这份备忘录，将它打印到一张 A4 纸上并折好，像履行教堂礼拜程序那样正式；然后交给了盖茨，以确保他们在微软公司的计划和战略上的观点是一致的。"查尔斯所写的内容综合了我们之前数十小时的讨论，"盖茨说道，"融合了我们两个人的看法。"

从当时的行业环境和微软公司的状况来考虑，这份备忘录实质上是一个言

简意赅的战略性纲要，一份很特别的文件；考虑到微软日后的作为，我们不得不感叹这份文件的非凡之处。那时，微软公司还是一家拥有不到 40 名员工的软件工具供应商，最为先进的产品是 BASIC 编程语言，即 Microsoft BASIC。它没有自己的操作系统，也没有像电子制表软件和文档编辑器那样的应用软件。

备忘录指出，微软公司应该生产面向“大众市场”的软件，而当时这种软件并不存在。在“战略”标签下，备忘录指出，微软公司应该拥有一个“完整的生产线”，具有一致的“商标名称”，而且微软旗下的产品都要相互兼容。

备忘录指出的一个关键战略目标是“掌控整个操作系统和编程环境”。结合当时的历史背景，这个目标更加凸显了盖茨以及像西蒙尼这样的元老级员工的长远眼光。1980 年 11 月，微软公司与 IBM 公司私下签订了一桩交易，那就是为 IBM 提供一个操作系统，或者主控系统，以确保这家大型计算机制造商在 1981 年进入个人计算机行业。但是，当微软与 IBM 签署这份重要合约的时候[10]，微软还没有自己的操作系统。因此，微软同意支付当地的西雅图电脑产品公司 2.5 万美元，来获取其 86-DOS 操作系统的再出售权。

1981 年 7 月，也就是 IBM 个人计算机问世的一个月前，微软公司又另外支付给西雅图电脑产品公司 5 万美元，彻底买下了该公司的 86-DOS 系统。这桩交易使微软公司在操作系统业务中站稳了脚跟。微软公司如何进入操作系统领域的故事已经成为微软传奇的一部分，人们耳熟能详。事后来看，这一事件清楚地反映出了盖茨采取的不光明手段，或者说他充溢的企业家天赋，每个人的看法各有不同。无论如何，西蒙尼于 1980 年 11 月写那份备忘录时，这个“掌控整个操作系统和编程环境”的战略性目标就反映出微软公司的预想；除了微软，当时没有一家公司能意识到这一点。

西蒙尼在 1980 年的备忘录中指出，应该在以下两个方面开展个人计算机软件业务：一个是办公市场，另一个是家庭市场。随后，他又写道：“‘型号年份’这一概念适用于软件吗？”他指着存放于家中的备忘录里的那句话，说道：“这看上去不正像 Windows 95、Windows 98 和 Windows 2000 吗？”备忘

录中关于办公应用的那一节指出，微软公司应该开发出一款能与 VisiCalc 程序相媲美的产品。“那就是 Excel。”西蒙尼说。Excel 是微软开发的一款电子表格制作软件。备忘录中还提到了对数据库的需要，也就是微软的产品 Access。此外，备忘录还提及了微软需要像 Bravo 那样的产品。“那就是 Word。”西蒙尼说道。他的备忘录还提到了 Macintosh 的前身——由施乐公司研发并应用在苹果第四代计算机上的 Lisa 鼠标点选型界面。西蒙尼写道，这种运算方式“经过花样翻新可以被运用到所有的微型计算机上，这将打开软件替换市场”。事实上，它的确能做到，譬如微软公司的 Windows 操作系统，目前 90%以上的个人计算机都安装了这一程序。

比尔·盖茨认为西蒙尼是“有史以来最伟大的程序员之一”。但是，盖茨看到的不仅仅是西蒙尼的技术天赋。他和西蒙尼志同道合，都认为软件在经济、文化和技术方面都与硬件不同。“我们当时的全部想法就是软件与硬件不同，只有按照这种想法，我们才能开公司，才能招聘到与众不同的员工，”盖茨回忆道，“查尔斯当时也有同样的认识，所以我们只要将查尔斯招入麾下，就相当于找到了一位组建公司的关键人物。”

1981 年 2 月，西蒙尼以高级产品研发总监的身份加入微软，负责应用软件的开发。“当时公司还没有任何应用程序，”他说，“我就是来做这件事的，负责技术部分。”西蒙尼加入微软时，微软正在研发一个与 VisiCalc 类似的电子表格应用软件 Multiplan；这是一个基于字符的产品，仅对 VisiCalc 做了一些适当的改进。这离西蒙尼的计划还有很大差距，他希望该电子表格中能加入一些生动的图片，有数据库功能，还要有其他一些与众不同的功能。然而，这些前瞻性的计划被暂时搁置。VisiCalc 一开始是专门为 Apple Ⅱ设计的，后来其他计算机也慢慢地运用了这个程序。

微软的战略是利用 VisiCalc 的缺陷，让其电子表格 Multiplan 能够尽快在

更多机器上运行。西蒙尼采用了一种聪明的编程策略来实现这个目标。微软的开发人员写出一种所谓的字节码，这些字节码处在软件的中间层，由程序人员进行编译，并不是在特定的机器上运行，而是在软件解释器上运行。这种思想通俗来说就是微软一次编写像 Multiplan 这样的应用程序，然后让厂商针对各自不同的计算机编写相应的解释器程序，将字节码转换成机器码。100 多家厂商与 Multiplan 签下了合同，其中包括 IBM、NEC、Olivetti、Datapoint、Commodore、德州仪器公司以及数字设备公司等。起初这看起来是一个不错的主意——在变革的时代，尤其对于微型计算机业，是一个前景广阔的策略。然而微软却为他这种兼容并包的方式付出了代价。Multiplan 运行很慢，特别是在一个新的竞争对手 Lotus 软件公司面前，这家公司将赌注全都押在了 IBM 的个人计算机上面。Lotus 的电子表格 1-2-3 在 IMB 个人计算机的微软操作系统上运行，有时也绕过微软系统来改进性能并添加一些功能。“Lotus 专门为一种机器——IBM 的个人计算机——编写 1-2-3 电子表格，”西蒙尼说，“它的运行速度比我们的快 5 倍，我们被 1-2-3 打败了。”

对微软来说有一个好消息：IBM 个人计算机当时已经成为行业标准，从而使得微软的 DOS 操作系统也成为业界标准的操作系统。因为 IBM 允许它的技术供应商们向其他公司出售产品，生产 IBM 克隆计算机的厂商开始建立并发展壮大，包括康柏、戴尔和捷威（Gateway）等。如果 IBM 公司不再约束整个产业，那么 IBM 标准技术的供应商——操作系统软件的微软和微处理器的英特尔——就会变得越来越强大。只是提供 IBM 标准的操作系统并不能确保微软取得成功，但是的确对微软帮助很大。随后，微软操作系统不得不进行一项复杂的转变，变成一个基于鼠标点选和图标的图形用户界面操作系统，简称 GUI（发音同 gooey，['gu:i]）。事实上，从 DOS 操作系统到 Windows 操作系统的转换很慢，有时候整个公司和个人计算机行业也不是很坚决，部分因为硬件也需要发展。然而在盖茨雇用了西蒙尼后，微软有了更长远的目标。西蒙尼是施乐公司帕洛阿尔托研究中心的代表人物，施乐公司帕洛阿尔托研究中心是图

形点选计算机的先驱。基于图形用户界面的施乐之星计算机推出之后，微软立即花重金买了一台。西蒙尼回忆说："这样我们就对它有切身的感受了。"

微软应用程序部门的大客户苹果公司也要求其加入图形计算功能；当时苹果公司是 GUI 的商业化先驱，在 1984 年推出了 Macintosh 计算机。Macintosh 从来没有真正威胁到 IBM 个人计算机标准的市场统治地位，但是它给用户带来了更好、更简单的个人计算机体验。在 DOS 操作系统的世界里，用户一开始看到的是黑色显示界面，上面只有一个白色的"C:>"，叫做"C prompt"或者"DOS prompt"。要访问一个文件，比如商业记事本或信件，用户需要知道这个文件存放在哪个计算机目录中，然后键入 DIR，空格，然后输入目录的名字。随后，要想真正将商业记事本或信件的内容显示出来，用户还要输入 CALL 命令，空格，再输入文件的名字。忘记任何一个目录的名字或者输入有任何错误都会导致查看失败。在 Macintosh 操作系统的世界里则截然相反，通过用鼠标点选图标或者屏幕上方的下拉菜单，文件的展示过程都是可视的。

对于普通的用户来说，使用 DOS 环境就像是在写程序。个人计算机从输入命令操作向点选操作转变是大势所趋。微软深知这一点，这个转变也会为公司的应用程序部门提供一个取代竞争对手的机会，这些竞争对手包括 Lotus 1-2-3、WordStar 和 WordPerfect，它们一度是行业的绝对领先者。"Macintosh 拯救了我们的应用程序，因为这是一个翻天覆地的变化。"西蒙尼解释说。

在软件业里，技术转移的主要方法是人员流动。Bravo 转变成 Word 的过程就是一个经典的例子。巴特勒 · 兰普森在 1995 年加入微软之后有机会窥探到 Word 的内部信息，很明显，Word 借鉴了 Bravo 的思想。兰普森指出："它们使用了相同的数据结构、相同的设计架构、相同的思路。"

作为一种工具，软件有很强的可塑性，因此有很多严谨、有力的法律定义来对其进行限制。而对于传递创意，将一些具体的概念和技术知识带到其他机

构中去的人员，限制却并不多。在西蒙尼去了微软公司之后，软件界的法律环境日渐趋紧；但是即便是现在，西蒙尼在设计 Word 时的借鉴以及对此前工作的拓展似乎也并不受限。西蒙尼说："Bravo 与 Word 之间有一种概念上的联系，而这两者的设计我都参与了。"施乐公司帕洛阿尔托研究中心并没有规定其研究员不得在其他机构从事相似产品的研究。在微软公司，标准的雇用协议上指明其程序员在离开微软一年以内不得在其他公司从事相同领域的研究。但是，西蒙尼其实遵守了这一规定，在设计 Word 程序之前，他在微软最初的工作是设计 Multiplan 电子表格程序。在当时的环境下，律师们仔细检查了西蒙尼的工作以确保他并未带来施乐公司的一字一句，如若不然，将会引起一些法律问题。当被问及关于 Word 文档编辑器的知识产权问题时，西蒙尼说："我们并没有借鉴施乐公司的任何文件。"然而，后来在翻阅一些文件的时候，他找出了一张施乐公司在 1980 年 5 月公布的 Bravo X 技术计划的复印件。"我可能不应该有这份文件。"西蒙尼边说边耸了一下肩。

第一版 Word 软件于 1983 年 11 月发布，远远超出了 IBM 标准个人计算机和打印机所驾驭的图表容纳能力。它的基础技术可追溯到施乐公司帕洛阿尔托研究中心的 GUI，之后被运用在 Macintosh 程序以及 Windows 系统中。微软的 Word 程序成为继 Windows 操作系统之后销量最高的软件程序，很多人用它来书写信件、商业报告、学习论文、新闻文稿、杂志故事和书籍。那时将自己关在一间小屋里希望写出奥斯卡最佳剧本或最佳美国小说的孤独艺术家们，创作杰作的工具极有可能是西蒙尼创建的 Word 文档编辑器。

西蒙尼愉快地翻阅着书架上那些新的娱乐性读物，都是一些关于美国水星、双子座和阿波罗航天任务的运行和程序指南，其中一些还有宇航员的签名。"我对签名不感兴趣，"他解释说，"我只想读那些指南。"西蒙尼的家是知识分子的乐园，有巨大的藏书室，现代艺术装修风格，工程学大事记典藏以及一间

计算机实验室；实验室洁白井然，从底到顶用白色木板装修而成，里面放着几个颜色鲜艳的豆袋座。（他说："这是帕洛阿尔托的传统。"）他参加的慈善活动大部分旨在推动严谨的知识追求；普林斯顿高级研究学院的数学大楼和牛津大学的一个荣誉教授职位都以他命名。前往牛津大学时，西蒙尼总是和知名的生物学家理查德 · 道金斯待在一起，道金斯同时也是一位"西蒙尼荣誉教授"。他们极少讨论商业、政治[11]，而是经常讨论科学、历史和文化。道金斯说，西蒙尼是一位"真正的知识分子，他对科学的方方面面，包括科学历史，都十分感兴趣，在很多方面都称得上知识渊博"。

在西蒙尼感兴趣的其他领域之中，西蒙尼对语言极为着迷。除了英语和匈牙利语，他还学习了法语、丹麦语和德语，时常邀请家庭教师轮流到他的家中。了解语言的作用及原理，对于思考编程语言和找出不同编程语言之间的差异是很有帮助的。"计算机语言与人类的语言很相像，"他说，"但是实质上，计算机语言和人类语言相比又受到许多限制。"西蒙尼说，编程语言并不像看起来那样通用，相较于人类语言来说更是如此。FORTRAN 是针对科学和工程的编译语言；COBOL 用于商业；C 语言用于系统编程；Lisp 则用于人工智能方面，等等。"看看其中的差异，"他说，"这和用德语做生意，用英语搞科研有所不同。"

自 20 世纪 90 年代中期以来[12]，西蒙尼一直致力于克服编程语言的单调局限性。他的雄心壮志是要变革编码书写技巧，这将使编程效率更上一层楼。很多人对此表示怀疑，他们认为西蒙尼的努力是不切实际的；软件工程的进步总是渐进式的。在过去的几年里，新工具和信息技术已经取得显著进展，但是，软件依然是一门近似于手工的艺术，这一点和硬件的发展极不相同。这些怀疑人士坚持认为，程序员可以期望的软件业最佳发展方式是渐进的，这在当时的计算机科学领域是一个盛行的观点。弗雷德 · 布鲁克斯在其编著的软件设计技艺局限性专著《人月神话》中，提出世界上根本没有能够提高软件效率方面的"银弹"。"软件行业很复杂，正是这种复杂性限制了我们。"

7

西蒙尼一心求变。他说："弗雷德·布鲁克斯所说的一切都是事实，但是在某种程度上并没说到点子上，因为现在的情况与20年、30年或40年前大不相同。"现在的计算机比10年前的机器要强大上千倍，这为设计新软件打开了一扇大门。他指出："没有银弹的说法是站不住脚的借口。"

西蒙尼一直在寻找一种技术，他希望可以藉此让开发者充分发挥智慧，或"意图"，不再局限于句法语言和惯例，这些句法可以说是个人计算机语言的围墙。他称这种系统为"意向编程"，这是一款高级编程工具，能够快速捕捉抽象思维，他们想让计算机做什么，计算机就做什么；用若干个代码模块对其进行控制，这些模块可以使用任意编码语言。"我试图将那些封闭的编程语言中蕴含的抽象概念解放出来，"他说，"然后让这些程序员把他们的聪明才智用在一个独立的交付工具上。"

意向编程系统很复杂，西蒙尼及其团队成员持有很多专利。从本质上来说，这一系统允许程序员在常规的编程语言之外工作，以便专注于思考自己到底想要实现什么（what），而不是如何去编写（how）。用计算机术语来说，what语句是声明式编程，而how语句则是程序性编程。传统的计算机语言要求程序员经常进行高密度的精细劳动。西蒙尼的新技术采用了精心设计的逻辑树，这个逻辑树与一个大型数据库相连，以便为软件中的数据提供合理的语境分析并进行合理的定义，也就是说，帮助推测数据背后的意图。没有想象力的计算机对于语境无识别力，西蒙尼认为这在很大程度上是由编程语言的固定性造成的。

在西蒙尼的计划中，使用一种编程语言编写的软件思想及特征可以与用其他编程语言编写的软件最佳特征混合在一起，而且还可以相互匹配，因而，通过意向编程开发过程的魔力，可实现软件之间的对话。意向编程是一种"转换"技术。就这一点来说，它使程序员可以越过常规语言的限制来开展工作。西蒙尼说："这是一种进步，超越了当前的编程范例，使软件设计更加接近人的意图。"

西蒙尼的这种想法能给软件生产率带来重大突破，抑或只是个遥远的梦想，只有时间能够告诉我们答案。如果能带来重大突破，将吸引更多的软件开发者使用微软技术，这将为微软带来巨大财富，并扩大其在业界的影响力。西蒙尼对此表现得很乐观。“如何节约和重复使用编程人员的智慧是计算机软件业最基本的问题。迄今为止，也是最好的问题……我想这迟早会变成一个大问题。我之前做过错事，但不是在这些大问题上。”

第 8 章

服务于大众的计算机：从 Gooey 到 Macintosh 的漫漫长路

高中时期，安迪・赫兹菲尔德因为当时计算机程序不够灵活智能而栽了个大跟头[1]。他为初中毕业生舞会编写了一个配对程序，结果却是一个女生与全班 1/3 的男生配对。对于这个十几岁的程序员而言，很明显一个女生不可能同时和 30、50 甚至 100 个男生约会；但对于不谙人情事理的计算机来说，这未尝不可，也就是说，要用非编程的方式来消除这种重复匹配的现象。

这次“舞会事件”中出现的程序缺陷并没有浇灭赫兹菲尔德对计算机的热情。早在一年前，当他在费城郊外的哈里顿高中参加编程课时，就有过一次挫折。他大部分的编程工作是使用 BASIC 语言，用遥控装置完成的。遥控装置

通过电传打字机终端与一台他从未见过的分时系统主机远程连接。在 20 世纪 70 年代，分时系统是高中生接触计算机的唯一途径，前提是学校资金足够充裕，能买得起价值数百万美元的远程主机上宝贵的机时。学校是按分钟购买机时的，赫兹菲尔德一人长时间独霸计算机的状况最终引起了校方的不满。他回忆说："学校禁止我使用那台机器。"后来他向校方认了错，并保证约束自己的行为，校方才慢慢对他解禁。但实际上，真正让校方原谅他的原因是，他向学校管理层证明了自己能够编写出有用的程序，例如用于计算年级平均分和每位学生班级排名的程序。

几乎从一开始，无论是解决数学难题还是设计约会方案，编程就在技术上和心理上深深地吸引着赫兹菲尔德。"你在做一样前所未有的东西，你对它有绝对的控制权。当你还是个孩子的时候，没有人会听你的，但是机器绝对听你的话，"当他回忆起控制机器的感觉时说，"这是一种与你的灵魂产生共鸣的东西。我爱它，而且我也善于控制它。"

赫兹菲尔德编程纯粹是出于兴趣。作为一个十几岁的少年，他从未想过有一天以编写软件程序为生。他认为程序员都是在公司混日子的人："他们工作糟糕，穿着大衣，系着领带，为银行或类似的行业编写程序。"这对于才华横溢的高中生来说毫无吸引力，他沉浸在当时的年轻文化中，其本质可以通过被其所排斥的越南战争和美国集团企业略知一二。赫兹菲尔德喜欢鲍勃 · 迪伦的专辑《重访 61 号公路》（*Highway 61 Revisited*），这个专辑歌颂了开放之路和"叛逆精神"。后来，他考上了布朗大学，学习物理、数学和计算机科学，但这段求学经历并没有改变他对专业程序员沉闷工作的印象。赫兹菲尔德回忆说："在那时的布朗大学，根本没有一个榜样，至少我一个也没有看到。"

赫兹菲尔德后来考入加州大学伯克利分校，学习研究生课程，但是学习热情并不高。因为他觉得这里的计算机课程枯燥、单调、扼杀想象力。1977 年，赫兹菲尔德偶然遇到了 Apple II，这才是他梦寐以求的计算机。这台机器在当时是计算机工程上的一大创举——紧凑的外形，1500 美元的合理价格，而且还

配置有彩色图形显示模式和微型计算机版 BASIC 编程语言。

Apple II 是个人计算机发展的第一道曙光。它的前景远远超出了当时书呆子式的、业余爱好者的机器——主要是装有微处理器的希斯套件；其中大多数具备高中水平工业艺术项目所需要的视觉吸引力。与之相反，Apple II 则具有引人注目的塑料外壳，是精益求精的苹果公司创始人之一史蒂夫·乔布斯对计算机美学的早期致敬。而令人惊叹的计算机内部配置——为达最佳性能而精心挑选、排列的芯片——显示出了另一创始人斯蒂芬·沃兹尼亚克的非凡工艺。赫兹菲尔德越深入研究 Apple II，越觉得惊叹不已。正如赫兹菲尔德所看到的，Apple II 的工程设计拥有个性，是个独立的个体，甚至带有一丝调皮——一种叛逆的精神。他回忆说："这是一台真正的计算机，但绝不仅限于用来更快地处理数据。"

赫兹菲尔德在 Apple II 的身上看到了他所追求的未来：编写程序，使个人计算机更加普及、实用并能为普通的计算机用户带来乐趣。他退学以后就开始为 Apple II 编写程序，他将精巧的设计融入软件当中，例如让计算机既能显示大写字母又能显示小写字母。1979 年，赫兹菲尔德加入了苹果公司，之后便参与编写 Macintosh 的核心程序。Macintosh 将点击式工作和虚拟桌面上的图形用户界面的理念带到了大众市场。Macintosh 团队中有神经学专家、生物化学家和图像设计师，他们同赫兹菲尔德一样，不断修正个人计算机中出现的程序缺陷，并坚信他们正在让世界变得更加美好。

Macintosh 为主流计算机领域带来了一种不同的观感体验——更友好的外观和更美的用户界面。设计计算机屏幕上软件布局的设计师必须在编程、图形设计、心理学和品位上作出决断。他们致力于用户和计算机之间最高水平的翻译和沟通工作：尽可能远离机器，更深入地了解用户。设计编程语言的计算机科学家也像在做翻译工作，但他们的受众面更窄，基本上是针对计算机专业人士。而用户界面设计者则在一个更宽广的平台上工作，他们必须要满足大众而非专业用户的要求。打个比方，编程语言设计者在和已经皈依的教徒对话，而

用户界面设计师正在劝导人们皈依。

用户界面软件与计算机科学领域的大多数其他软件不同，它需要不同的设计技巧与思维。比起工程学来说，这更像一种人文科学，因此需要不同种类的人才，比如像安迪 · 赫兹菲尔德这样深切关注观感体验的编码黑客。他解释说："在要求技术性、准确性和客观性的计算机，与带有不确定性、感性和主观性的人类之间徘徊，真的很有意思。我一直很喜欢艺术，尤其是文学和音乐，而且我觉得人文因素能够将工程学提升到艺术的领域。"

在 Macintosh 面世大约 15 年后，赫兹菲尔德回归了。1999 年，赫兹菲尔德与当年 Macintosh 团队的一些成员共同创建了 Eazel 公司，主要业务是为使用 Linux 系统的计算机设计用户界面。Linux 是一种基于 Unix 系统、使用开放源码模型的操作系统，通过这种模型可以合作开发并免费分享软件。由于金融市场不景气，投资者纷纷躲避投资风险，Eazel 公司在 2001 年 5 月宣告失败。赫兹菲尔德非常失望，但是这位矮小、精明、带眼镜的"软件魔术师"（从苹果公司开始，他的名片上就只印这一个头衔）对于普及个人计算机的信念从未动摇。"对我来说，这不是一项工作，而是一种召唤，为坐在计算机前面的用户提供更美妙的体验。"

对更加人性化计算机和软件的真正探求始于 J.C.R.利克里德。[2]这位哈佛大学培养出来的心理学家于 1950 年在麻省理工学院启动了一个心理学项目。他决定将这一项目用于电气工程系，希望工程师在做设计时能够时刻将人文需求牢记于心。20 世纪 50 年代，他效力于美国国防部的 SAGE（半自动地面防空警备系统）项目和其他将雷达与早期计算机相结合的防空监控系统。在这种系统中，快速清晰的信息显示至关重要，利克里德负责设计显示控制台。他说："有两个信息显示方面的问题：建立系统使之工作和设计一个好的用户界面。"后来利克里德来到位于马萨诸塞州剑桥市的 BBN 技术研究公司工作。1960 年，

利克里德写了一篇具有开创意义的论文《人机共生》。

在论文中，利克里德提出计算机应当发挥的功用是提高知识工人的生产力，提高人类智力，而不是取而代之。这个论点看似有目共睹，然而在当时，一大批人工智能的研究者都相当乐观地相信，计算机能够在不久的将来在思考和解决问题领域与人类智力相媲美，甚至超越人类智力。利克里德说，也有一天会吧，然而计算机最大的益处在于能够协助人类做更多的事情，它们去做那些单调的工作，而使人们能够解放出来，真正地去思考。基于利克里德的研究，有大量需要计算机去做的单调工作。他估计，技术工人 85%的时间都在做耗时耗力的“文书性或机械性”工作。他所提出的“人机共生”理论就是寻找一种方法将这 85%的工作交给计算机来完成。

但是，人类与计算机关系进一步发展的道路也并不平坦。与计算机交流就是个首要问题。“人类语言与计算机语言根本不同，”利克里德写道，“这也许是人机共生的最大障碍。”他看到了如 FORTRAN 和 Algol 等编程语言的进步与发展，但这都是人类向计算机妥协的结果。“通过采用易于翻译成计算机语言表达法的标准格式，人类证明了自己的灵活性，”他写道，并且指出了这种方法的局限性，“但是，为了达到人机实时合作的目的，采取另一种截然不同的交流和控制原则是绝对有必要的。”他接下来又讨论了提高显示和输入设备性能的必要性，甚至还提及了手写及语音识别等未来技术。

利克里德的论文本身就是对计算机领域研究的一项重大贡献；而不久之后，他就对这一领域产生了直接影响。1962 年，他受聘于美国国防部先进研究项目署，负责其新成立的信息处理技术办公室。该研究署接受了数百万美元的资助来进行计算机研究，而资助者自然而然地对人机互动产生了特别的兴趣。利克里德的兴趣与研究都来自他在防空系统和 BBN 公司工作时操作计算机的实际经验。BBN 公司购买了美国数字设备公司 1959 年出产的第一台计算机 PDP-1。

利克里德提出，知识工人仅有 15%的时间用于真正意义上的思考，这一理

论源自一个夏天他对自己工作习惯的记录与研究。他长时间与计算机打交道，直觉告诉他，计算机确实能够改变世界。“我认为，这将是对人类的思考方式和工作方式的一次革命。”[3]1988 年，在利克里德离世两年前，他回忆说：“我每天坐在计算机前四五个小时，甚至更长时间。当时这样工作的人非常少，而我就是其中一个。这样的强制性劳动让我对当时设备的局限性感到很沮丧，但同时我也看到它在飞速地向好的方向发展。”利克里德早期就坚信信息技术会在提高生产率方面产生巨大的经济效益。1988 年，在提到计算机尚未实现的潜力时他说：“我觉得我们能让生产率增加 2 倍、3 倍、4 倍、10 倍，甚至更多；而且现在这一看法依然未曾改变。”

1962 年，就在利克里德来到美国国防部先进研究项目署不久，他收到了人机互动领域中与他持有相似观念的学者道格拉斯 · 恩格尔巴特的一封来信。当时，恩格尔巴特是斯坦福研究院的研究员，刚刚发表了一篇学术论文《增强人类智力：一个概念框架》[4]。作为对利克里德的回应，他提出了知识工人面对信息过剩和日趋复杂性的困境，并且声称计算机是将人类从中解脱出来的“最及时的承诺”。

在论文中，恩格尔巴特勾勒出了工作者未来在计算机的协助下工作的情境。只需一个键盘和一个高级的显示控制台，我们就可以操控文字和符号信息。在屏幕中键入文件的初稿之后，工作者就可以“编辑、改写、编译和删除。这很有意思——‘把那句话放回到这两句中间’——一眨眼的功夫就完成了。”完全可以抛弃使用铅笔、打字机、剪刀和胶水的繁复体力劳动，取而代之的是用计算机“编辑文本的新方法”。在计算机上，“通过对初稿的重排、摘录以及键入新的词语和段落，就能够快速编辑出第二稿……你可以更加容易地整合新的想法，让创造力更具连续性”。未来的知识工人会对这一变革惊叹不已。恩格尔巴特甚至设想了工作者们兴高采烈的情形。“你会觉得这个灵活的、具有

试验性的方法真的符合你想表达的思想。天哪，你可以写数学公式、广告或者一首诗，样样都行。”他强调，任何一个辛苦工作的“富有创造性的问题解决者”，借助计算机都会取得事半功倍的效果。

40 年后，我们很难充分评价恩格尔巴特预言的准确性。他的预测变成了现实，计算机已经进入寻常百姓家。但是不要忘记，当他做出以上预言时，计算机价格昂贵，需要几十万甚至几百万美元，每个人拥有一台个人工作站的理念似乎是个遥不可及的梦想。当恩格尔巴特发表论文时，仅显示器的价格就在 2~6 万美元之间。恩格尔巴特 1962 年对未来的预想也并不完全准确。例如，他曾设想未来的工作者会像使用键盘一样使用光笔。但实际上，光笔并没有成为操控屏幕上字符和图形的最普及输入设备；而真正获得成功的是恩格尔巴特几年之后发明的鼠标。

在政府的资助下，恩格尔巴特和他斯坦福研究院的研究团队在 20 世纪 60 年代做了一系列的创新，改变了人们与计算机的互动方式。除了鼠标以外，另一个重大发展就是能够在屏幕上显示多个数据视图，就像并排的“窗口”。窗口的概念颠覆了人与信息的关系。计算机窗口摒弃了每页显示信息的方式，像镜头一样展示数据视图，也就是我们后来熟知的计算机空间。而另一些窗口可在屏幕上显示其他的视图或者视角。恩格尔巴特的发明先后被施乐公司的帕洛阿尔托研究中心、苹果公司和微软公司所采用、润色和修改。但是现代计算机显示方式还是始于恩格尔巴特的，因此施乐公司帕洛阿尔托研究中心的负责人阿兰·凯伊称恩格尔巴特为“圣经里的先知”。

恩格尔巴特硕果累累的事业始于费城。[5]恩格尔巴特 20 岁的时候还只是个海军无线电技术员，在菲律宾的莱特岛上等待二战结束后归国。他漫无目的地走着，不知不觉地来到一座高脚楼，这里是个临时搭建的红十字会图书馆。在浏览杂志的时候，他无意中看到了 1945 年 7 月《大西洋月刊》上范内瓦·布

什的文章《成若所思》。范内瓦·布什是为二战服务的科学发展研究所负责人。随着战争的结束，布什指出了技术所面临的新挑战，该挑战来自于“堆积成山的研究”。为了解决这一问题，他提出了 memex 的概念。memex 是一个带有显示器和键盘的机械工作台，能够在微缩胶片上储存一个人所有的书籍、录音、杂志和其他通信信息等资料。按照布什的预想，储存实质上可以是无限的。“如果一个用户每天输入 5000 页资料，那么他需要几百年才能把这个知识库填满。”恩格尔巴特读到这篇文章时激动万分。他回忆说：“有人在思考这样的可能性，这让我浑身为之一震。”

但是直到几年之后，恩格尔巴特才将这振奋人心的想法付诸行动。当时他在位于加州山景城、政府下属的艾姆斯实验室做工程师，研究风洞试验，但是他并不甘心于此。考虑到未来的发展，恩格尔巴特说：“我找到了我的职业方向，我希望将剩余的精力都投入到最大限度地改善人类命运中去。”回想范内瓦·布什，再展望未来，恩格尔巴特选择了投身于“增强人类智力”。1951 年，他进入加州大学伯克利分校攻读电气工程博士学位，并在 1957 年加入了斯坦福研究院。他早期向政府申请研究资金的努力并未见成效，这与美国东海岸在当时占研究界统治地位不无关系。恩格尔巴特回忆说，1961 年的一封拒绝信上这样写道：“鉴于你所感兴趣的研究极其先进的编程支持，而你所在的帕洛阿尔托地区远离计算机专业研究的中心地区，因此我们认为你不能为项目研究配备足够的人员。”

利克里德于次年来到美国国防部先进研究项目署的时候，资金问题开始出现好转。到了 1964 年，斯坦福研究院开始对能够在屏幕上进行标记和移动的设备进行实验。研究所内部进行了一次小型市场样本的严格筛选，鼠标独占鳌头。鼠标战胜了光笔、操纵杆以及其他一些不常见的设备，如附着在使用者膝盖上的设备和“鼻子指向控制”装置。恩格尔巴特解释说：“鼠标因为能够快速、准确地在屏幕上对文章做出选择而击败了其他设备，大获全胜。在几个月的时间内，我们还把其他几种设备留在工作台上，以供用户自行选择合适的设

备；但当看到每一个人都选择鼠标的时候，我们就放弃使用其他设备了。”

恩格尔巴特的成就为世人所接受是个缓慢的过程。1968年，斯坦福研究院在旧金山出席了一次大型计算机会议，并在会议上就他们的工作站技术做了一次令人印象深刻的展示。恩格尔巴特站在台上，将工作站的影像投影在一面20英尺的屏幕上。他演示了如何使用鼠标操作互动程序，将屏幕分成多个由文字和图像组成的窗口。90分钟的展示就像是业界名家的技术演示，密密麻麻的天线和微波将会场与30英里以外的斯坦福研究院实验室相连。“会议结束后，我们都信心满满地以为全世界都会开始谈论增强智力了，”恩格尔巴特回忆说，“可是，事情并非如我们所料。”

多少年来，恩格尔巴特经常说起，他十分惊讶于斯坦福研究院的工作成果过了如此之久才为计算机界所接受。思维惯性和想象力缺失无疑起到了一定的作用，但经济发展和人性本身的原因也存在其中。当时，计算机的造价依然昂贵，因此个人工作站的概念似乎还只停留在理论层面——令人着迷但尚不实际。1969年，在斯坦福大学读研的唐·钱伯林参加了恩格尔巴特的讲座，他后来开发出了 SQL 关系数据库语言。当时，恩格尔巴特用鼠标操作一艘船的图形在工作站的屏幕上来回游动。钱伯林回忆说：“这很棒也很有趣，但我没想到20年或25年后，每一个西方人都把一天中的大部分时间用在摆弄鼠标上。”

在某些方面，恩格尔巴特正在引领的并非是温和的演变，而是一场不容妥协的革命。他的“增强”工作站是个全能的系统，其中某些元素更易为大众所接受。在用尽政府资助、离开斯坦福研究院之后的几年中，恩格尔巴特在研究院发明的键盘设备依然长盛不衰。他一手持鼠标，一手持键盘作为鼠标的辅助性输入工具。五个键可以像和弦一样敲击，如同钢琴，使用者只需学会一种类似速记的简洁代码。恩格尔巴特指出：“无论何时，当你听到有人说要‘简单易学，运用自如’时，你都要标个问号，质疑一下。什么叫做自如？自如就是已经被大家逐渐所接受的。”

对于恩格尔巴特来说，追求简单易学、方便使用永远是他的第二选择。他用三轮车举例，骑三轮车比骑自行车好学，但是一旦经过训练，学会了骑自行车，那么其速度和适用的地形都不是三轮车可比拟的。恩格尔巴特并不羞于要求人类屈服于机器。斯坦福研究院的鼠标就有三个按键。

施乐公司帕洛阿尔托研究中心的人机交流研究明显倾向于屏幕前的使用者。[6]20 世纪 70 年代，帕洛阿尔托研究中心创造了与计算机互动的现代手法——一个排列有小型“图标”的“桌面”、一个鼠标、叠搭的窗口和一个简化的“菜单”来引导用户使用复杂的计算机。他们设计的计算机基于“图形用户界面”（Graphical User Interface），简称为 GUI（发音同 gooey，['ɡu:i]）。它还有另一个由首字母缩写构成的名字 WIMP，代表其主要组成元素——窗口（window）、图标（icon）、鼠标（mouse）和下拉式菜单（pull-downmenu）。那些认为小图片在计算机实际应用中毫无用处的人们觉得这个名字真是再适合不过了①。当然，这些反对者最终不得不屈服，看着图形用户界面在 20 世纪 80 年代登上了历史舞台。

施乐公司于 1970 年在帕洛阿尔托成立了研究中心。由于复印机业务的资金运转良好，施乐公司能够负担计算机研究中的开支。但是管理层认为，价格低廉的日本复印机已经进入小型商用机市场，这会对公司一直盈利的特许经营方式构成威胁。在这种理念的驱动下，施乐公司希望从长远考虑，除了复印机以外，使产品更加多元化。而帕洛阿尔托研究小组的使命则是寻找“未来的办公室”。对未来的追求源于利克里德倡导的人机互动思想。施乐公司帕洛阿尔托研究中心的研究员，大部分来自美国国防部先进研究项目署资助的各个项目。

① WIMP，意为“懦弱的人，无能的人”。——译者注。

在20世纪70年代的施乐公司帕洛阿尔托研究中心，阿兰·凯伊是用户界面研究的智慧之光和精神导师。他聪明、叛逆，曾因为抗议学校对犹太人的名额限制而被学校开除；有一段时间，他做过专业吉他手。之后，在一次空军派遣工作中，他显露出了在计算机编程方面的天赋。退役之后，他重返校园，在科罗拉多大学学习数学和分子生物学。他之后又考入犹他大学研究生院，并对计算机产生了浓厚的兴趣。20世纪60年代后期，犹他大学成为美国国内领先的计算机图形图像研究中心。这也是美国国防部先进研究项目署扶持的研究项目，由大卫·埃文斯和伊凡·苏泽兰负责。

1966年，阿兰·凯伊刚刚来到犹他大学，就收到一本苏泽兰的经典著作《画板：人机图形通信系统》。画板是20世纪60年代初苏泽兰为早期图形显示计算机TX-2设计的程序。苏泽兰是麻省理工学院的研究生，画板是他的博士研究项目，这一项目在当时是个划时代的进步。在人们普遍认为计算机仅仅是大型计算器的年代，画板是第一个绘图程序。而苏泽兰认为自己是个"图形思考者"，他曾解释说："如果我能够图像化一些可行的解决方案，就更有可能找到最好的那一个。"为什么不借助计算机进行图形思考呢？有了画板，人们就可以使用光笔绘图、做设计和制作蓝图。这个程序能让用户修改、复制并储存图像，还能在显示器上放大和缩小图像——以纸张、铅笔和橡皮等传统媒介所无法企及的方式，尝试并检验用户的设计。"画板能做的事情非比寻常，"阿兰·凯伊回忆说，"它与我之前所见过的任何一种计算机应用都截然不同。"[7]这是现代交互式计算机图形的伟大发明。

几乎在接触画板的同时，阿兰·凯伊也接触到了另一项产生重大影响的发明，这就是由挪威人克利斯登·奈加特和奥利–约翰·达尔设计的模拟编程语言Simula。Simula由不同数据类型的"类"组成，并吸收了继承等概念。Simula能够使程序员分别标注不同种类的数据，按照逻辑层级组织数据，然后使信息

在一个数据类与另一个相关联的数据类之间流动。这个系统简化了编程，但比其他大多数的编程语言更加灵活。Simula 促使阿兰・凯伊用生物学方式思考，比如细胞代谢，包括“简单的机制控制复杂的过程；能够分化成各种所需结构单元的某种结构单元”。

正如画板是一种全新的计算机应用一样，Simula 也为编程语言提供了一种不同的方法。阿兰・凯伊解释说：“毫不夸张地说，我在此后产生的大部分新想法都源于 Simula。它能为组织计算提供一种全新的方式，其前景让我痴迷。”

次年，道格・恩格尔巴特来到犹他大学展示他的系统。对于阿兰・凯伊来说，恩格尔巴特的想法是“交互式计算应该具有的样子，我立即采用了他的很多观点”。同时，阿兰・凯伊也开始思考摩尔定律，该定律是英特尔创始人之一戈登・摩尔于 1965 年提出的——集成电路芯片上所集成的晶体管数目大约每 18 个月翻一番。摩尔推论，如果这一趋势继续下去，计算机的处理能力将出现指数级增长。对阿兰・凯伊来说，摩尔定律所暗含的内容一下子清晰了起来——当时笨重而昂贵的计算机将会变为家家负担得起的台式计算机，甚至笔记本计算机。这只是迟早的事，何况还有其他因素的影响。

对于阿兰・凯伊来说，这是一个启示。“现在的计算机不会存在太久……计算机领域终将出现如人类第一次接触哥白尼的日心说并重新思考天地关系时出现的困惑与迷惘。”阿兰・凯伊于 1967 年预言的哥白尼式变化出现了：随着 IBM 在业内崛起并占领统治地位，大型终端机注定要失去计算机领域“宇宙中心”的地位。“全世界共有不过几千台大型终端机，都由公共机构控制”，他预测，终有一天“它们将被数以百万计的个人计算机和用户所取代，而且多数不受公共机构的直接控制”。

在接下来的一年中，阿兰・凯伊也受到了其他思想的影响。在美国国防部高级研究计划局为其研究生提供的住处，阿兰・凯伊参加了马文・明斯基的一场讲座；明斯基就职于麻省理工学院的人工智能实验室。在讲座中，明斯基抨击了死气沉沉、因循守旧的传统教育模式。在这里，阿兰・凯伊第一次听到

瑞士哲学家、心理学家让·皮亚杰关于儿童童年期发展和自我发现的理论，以及麻省理工学院的西摩尔·派珀特使用为儿童设计的编程语言，对“从实践中学习”进行的研究。阿兰·凯伊拜访了派珀特并且观摩了他的儿童实验。在实验中，儿童利用一种叫做 Logo 的编程语言及环境来构造事物并解决问题，其中包括操纵一个类似海龟的图形在屏幕上运动。阿兰·凯伊阅读了马歇尔·麦克卢汉的《理解媒介》，并仔细思考了其隐晦的名言：“媒介就是讯息。”他在访问伊利诺伊大学时看到一块 2.54 cm 见方“带有氖气的玻璃板，上面的每一个小点都可以按照指令亮起——这是最早的平板显示器”。这种装置逐渐发展成为现在数百万笔记本计算机上所使用的显示器。

这些新想法渐渐在阿兰·凯伊的脑海中成型，形成了他对个人计算机的概念。恩格尔巴特把他的系统想象成个人“车辆”，即与 IBM 的大型机“铁路系统”相对的个人小汽车；而阿兰·凯伊则认为个人计算机应该是个人的互动式媒介。这一观点包含一系列内涵，其中包括人们学习使用计算机的方式。“对于车辆，人们要等到高中才能学习驾驶；但是对于媒介，人们可以从童年时期就开始学习。”阿兰·凯伊很快就将想法付诸行动。他把自己设想的个人计算机用纸板做了一个模型。想起在印刷机发明几十年之后，奥尔德斯·马努蒂尔乌斯出于将书装进鞍囊的考虑而规范了现代书籍的规格，阿兰·凯伊从计算机使用者的角度，细致考虑了他梦想中的计算机规格。这种计算机要像书籍一样便于携带，可以放置于膝上；要有平板显示器、键盘和一支光笔（因为它有手写识别功能）。他随后在模型中灌入小铅粒来控制计算机的重量（不到 2 磅）。计算机将通过无线网络交流。阿兰·凯伊将这款概念计算机称为 Dynabook。

Dynabook 就像查尔斯·巴贝奇的分析机，在计算机发展史上占有重要的地位，却从未被真正地制造出来。尽管如此，Dynabook 以阿兰·凯伊所描述的方式在计算机界产生了深远的影响。在计算机行业以外，Dynabook 并不为人所熟知；但是对于业内人士来说，它长期以来都是个人计算机发展的理想目标。计算机公司依然在追寻阿兰·凯伊 1968 年的设想，并在一步步地接近目标。

一台机器并不是媒介，梦想中的机器也是一样。在阿兰·凯伊的设想中，服务于大众的计算机意味着软件要改变计算机的用户体验。他解释道："数百万的潜在用户意味着用户界面要变成一个学习环境。"对于阿兰·凯伊来说，一个全新的软件环境表示一切从头开始——新的编程语言。计算机语言的问题似乎在于使用过多的机器术语，这些生硬的程序、数据结构和函数缺乏自然语言的灵活性。因此，阿兰·凯伊开始着手设计一种称为 Smalltalk 的新语言。20 世纪 70 年代，随着 Smalltalk 在施乐公司帕洛阿尔托研究中心逐渐成型，它已经不仅仅是一种编程语言，还是一种组织计算的新方式，用更加灵活的方法将人类的问题呈现在机器中。

阿兰·凯伊摒弃了传统编程语言机械、死板的结构，采用生物学体系，其中的基本构成模块是"万能细胞"或对象。[8]任何事物——数字、词语、列表和图片——都可以用对象来表示，它们通过发送信息相互连接。Smalltalk 的每个对象本身都可以被视为自成一体的虚拟计算机。"我想到，如果只针对虚拟计算机来编程，那么过程将容易得令人难以置信，"阿兰·凯伊回忆说，"就好像生物体仅仅由细胞构成一样。"他还将 Smalltalk 看成一个软件环境，非常像早期的互联网。美国国防部资助的阿帕网（ARPANET）当时正在成型，"大量的计算机只通过相互发送信息来交流"。

阿兰·凯伊将这种细胞式的构建软件方式形容为"面向对象的编程"。其他语言也使用了对象技术（早期的 Simula 以及后来的 C++和 Java 语言），但从概念上看并不纯正，也并未与 Smalltalk 一样坚持对象哲学。阿兰·凯伊称，Smalltalk"依然是唯一真正面向对象的编程语言"，系统中的每一个对象都是实实在在并且自成一体的。现在主流的"面向对象"编程语言，C++和 Java，都过于注重实际效用，为了追求速度和适应程序员的习惯而做了妥协。对于阿兰·凯伊这位计算媒介中大胆的概念派艺术家来讲，这样的妥协未免有些目

光短浅。“他们常常只追求解决一部分问题，为了所谓的符合‘实际’而沾沾自喜。”令他感到自豪的是，Java 和微软互联网软件.Net 的发展在某种程度上效仿了 Smalltalk。互联网迫使计算机业界效仿 Smalltalk 的细胞式设计，依据化繁为简的原则进行思考。他指出：“大体上讲，软件正在慢慢向 25 年前的施乐 Smalltalk 靠拢。”

与施乐公司帕洛阿尔托研究中心的 Alto——阿兰・凯伊称为 interim Dynabook——同时问世的图形用户界面结合了包括图标、窗口和鼠标在内的早期研究成果。正是因为他所在团队的关注点在孩子身上，才将这一切结合起来，“形成强有力的理论依据和经得起时间考验的样板”。他说：“在早期，这就让用户界面的目的来了个 90 度的大转弯，从‘普及功能性’变成了‘创造能够从实践中学习的使用环境’。”

梦想家阿兰・凯伊的身边聚集着一群富有想象力的帮手。他招募了阿黛尔・高德伯格，她是一位教育技术学专家，也是位经验丰富的程序员（Smalltalk 的共同开发者之一）。她与阿兰・凯伊为研究出谋划策，并在帕洛阿尔托地区 12~15 岁的学生当中进行理论试验。将不同用户界面的构想编写成程序的任务通常由丹・英戈尔斯完成，他是 Smalltalk 的主要开发者。还有其他许多人做出了卓越贡献，包括拉里・特斯勒，他为查尔斯・西蒙尼的 Bravo 设计了图形界面。施乐公司帕洛阿尔托研究中心的学术气氛融洽，不同的研究团队的科学家经常相互合作。计算机科学实验室（西蒙尼、巴特勒・兰普森、查尔斯・萨克尔等人的工作地点）的研究员会定期与阿兰・凯伊学习研究小组的人交流。而用户界面这一创举，毫无疑问来自于阿兰・凯伊的团队。这是他们长期观察人们如何使用计算机，并不断做出改进的结果。

兰普森回想起特斯勒为文字处理软件 Bravo 设计用户界面之前的情形。[9]“并不是我们没有考虑到用户界面的重要性，而是我们在计算机科学实验室里

根本就没有资源，”他说，“用户界面设计是一项非常艰难的重复性劳动。构思很简单，但要将这种想法在许许多多的用户身上试验……关键在于不能让字节显示出来。要是显示出来，从某种意义上来讲，你就搞砸了。”

20 世纪 70 年代后期的 Alto 系统几乎不显示字节。很多计算机科学方面的进步都是渐进式的改变。当年曾在施乐公司帕洛阿尔托研究中心工作的约翰・索殊说，很少会有“看到某些东西情不自禁发出赞叹的时刻，但是 Alto 的用户界面做到了。对计算机有点了解的人第一次看到它时，都会惊呼：天哪！”[10]

施乐公司帕洛阿尔托研究中心的成果对苹果公司的一些人产生了影响。史蒂夫・乔布斯参观研究中心的故事尽人皆知。其简短的版本是，1979 年 12 月，乔布斯及其团队来到这里，看到图形用户界面的“圣火”，就将其带回了苹果公司。但实际上，背后的故事并非如此简单。其实在参观前不久，苹果公司内部的一些人员就已经获悉施乐公司帕洛阿尔托研究中心的工作成果，其中包括杰夫・拉斯金。[11]杰夫・拉斯金博学多才，拥有计算机科学和哲学学位，曾任视觉艺术教授，还是旧金山一家小型歌剧院的指挥。拉斯金曾经动员苹果公司的工程师参观帕洛阿尔托研究中心；实际上早在那次访问之前，正是拉斯金在 1979 年 9 月启动了 Macintosh 项目。

参观帕洛阿尔托研究中心的确转变了苹果公司其他人的想法，在此后几年中帮助他们形成了用户界面的设计方案。首先是 1983 年推出的 Lisa，随后是第二年上市的 Macintosh。比尔・阿特金森在参观之后颇为钦佩，后来他与离开帕洛阿尔托研究中心加入苹果公司的拉里・特斯勒一起负责 Lisa 的设计。实际上，Lisa 与后来的 Macintosh 在图形界面上有很多共同之处。这两个项目在开发在时间上有重叠，因此常在苹果公司内部相互争夺人力和资源。所以安迪・赫兹菲尔德等人把为 Macintosh 编程的任务形容为，将 Lisa 的用户界面

“强塞进”一个小巧的、大家能够买得起的机器里。

苹果公司本来打算倚靠 Lisa 大力拓展商用计算机市场，但是其售价超过 12 000 美元，而且在 Lisa 面世的两年前，IBM 就已经开始向商用客户销售比 Lisa 便宜得多的计算机。虽然与 Lisa 相比，IBM 的个人计算机显得过于老旧，但其低廉的价格更具吸引力，也得到大多数商务人士的认同，因此它的市场地位无法撼动。Lisa 在技术上成功、市场上失败，就如同当年的施乐之星。

在 Lisa 问世的几年前，苹果公司高层掌权者将乔布斯排挤出了该项目。年轻的史蒂夫・乔布斯是个难以捉摸的完美主义者，他思想活跃、情绪多变，因此被看做不安定因素。到了 1980 年，他终于接触到了 Macintosh，并在第二年挤掉拉斯金，开始负责该项目；拉斯金随后离开了苹果公司。Macintosh 在 1984 年问世时，已与拉斯金脑海中的规划大相径庭。当时拉斯金仅把 Macintosh 设想为一台没有鼠标、售价为 1000 美元的“信息工具”。而现在的 Macintosh 更像是精简、改良版的 Lisa，售价为 2495 美元。这一改变最大的赢家就是乔布斯。

一批有艺术气质的工程师使 Macintosh 诞生于世。安迪・赫兹菲尔德设计并编写了 Macintosh 用户界面的工作引擎——显示器控制软件。他在机器逻辑板上单一的普通 ROM（Read-only Memory，只读存储器）芯片上编码。将 Macintosh 的观感设计融入如此有限的内存空间里，可以说是一项精巧的编程艺术——由代码写成的俳句，Lisa 1/8 的内存和计算机硬件的局限性迫使他做出这样的创新。

为了尽可能地取 Lisa 之精华，赫兹菲尔德为每一个字节的内存和每一个循环的性能精打细算。大部分的 Lisa 用户界面软件是用 Pascal 语言编写而成的，然而赫兹菲尔德从更加贴近机器的角度出发，选择了一种适合 Macintosh 的微处理器的汇编语言来编写程序——摩托罗拉 68000。作为一种高级编程语言，

Pascal 语言在编译器外还要求额外的软件来将指令翻译成机器编码。赫兹菲尔德抛弃了这种软件自动操作，就好像飞行员在特殊天气条件下不再依赖自动驾驶模式一样。他看着 Pascal 编译器的工作结果，就是从 Lisa 翻译过来的编码，认为自己能做得更出色。他说："我发现，如果用 68000 汇编语言手动编码，我能让它缩小一半以上。"

这种大刀阔斧的削减是通过各式各样的编程窍门来实现的。例如，赫兹菲尔德注意到 Pascal 生成的标题（headers），即每个编程任务开头的陈述，是可以省略的。每个编程任务末尾出现的结尾（trailers）也可以省略。他还将一些数据从内存中转移至名为寄存器的微小高速电路当中，以此扩大内存空间。通过一步步改进，Macintosh 逐渐成型了。他解释说："我用汇编程序能做得更好，因为我一直以来都在和实际的计算机资源打交道，而非纸上谈兵。"赫兹菲尔德指出，像这样手动编程取得成果，如今恐怕很难实现了。编译器不断改进，微处理器也变得越来越复杂，能同时执行 4 个指令。他说："所以现在的编译器在大多数情况下，能比人类做得更出色。"

在某种程度上，在某种程度上，赫兹菲尔德认为自己是比尔·阿特金森 Lisa 工作的技术成果受托人。赫兹菲尔德说："我把比尔的工作成果重新包装，使其进入一个拥有数百万人的用户市场。"盖·特里布尔是阿特金森多年的好友[12]，后来成为 Macintosh 软件团队的负责人，并与赫兹菲尔德共同组建了 Eazel 公司。特里布尔和阿特金森同在加州大学圣地亚哥分校学习，并在计算机中心结为朋友。他们共同进行了一系列计算机项目，其中有一部计算机图形影片，让观影者能够体验一次在人类大脑内部的虚拟飞行。他们的一个项目还得到了美国国家科学基金会的资助，其照片刊登在一期《科学美国人》杂志的封面上。后来，特里布尔和阿特金森又同时进入华盛顿大学进修——特里布尔在医学院学习神经学，阿特金森学习生物化学。对于这两人来说，计算机只是追求科学的工具；但是，到了 20 世纪 70 年代中期，微型计算机的问世预示了个人计算机的未来，特里布尔和阿特金森的观念发生了转变。阿特金森首先顺

应潮流，在 1978 年加入了苹果公司；两年以后，特里布尔也加入了苹果公司。赫兹菲尔德在进入苹果公司的最初几年跟着阿特金森一起工作。“我是比尔·阿特金森培养出来的，”20 年后，赫兹菲尔德回忆说，“他有一条准则：让用户满意。”

但 Macintosh 并不仅仅是缩小版的 Lisa，它的用户界面与 Lisa 有一些很重要的不同。Lisa 的设计不是以启动和退出应用程序，而是以打开和关闭文件为中心的。Macintosh 团队中的史蒂夫·卡普斯指出：“它是 100%以文件为中心的。”[13]例如，Lisa 没有 Macintosh 中用来启动应用程序的“苹果菜单”。而且 Lisa 的文件都是用文字处理器和苹果公司的 LisaWrite、LisaDraw 和 LisaProject 等程序制作而成的。Lisa 与世隔绝——用苹果公司自己的软件；而且是个给刻板的办公室环境使用的严肃计算机，不带有计算机游戏。

与 Lisa 截然相反，Macintosh 是个开放的平台。它为外部供应商的软件程序提供了用武之地——图形、游戏、文字处理等任何一种软件开发者想要制作或投入市场的软件。因此，用于启动应用程序的用户界面对 Macintosh 来说至关重要。依赖外界的软件公司是有很大风险的。市场上的应用软件增速缓慢，致使 Macintosh 在上市初期销售低迷。史蒂夫·乔布斯是最大的受害者，他在 1985 年被逐出了公司。但是，建立更加开放的平台显然是正确的策略，最终使得苹果公司的 Macintosh 和数百家第三方应用程序供应商建立起了相互依存、互惠互利的纽带。这种模式重复了当年 Apple II 的成功商业生态系统；然而真正使这种商业模式日臻完美的不是苹果公司，而是只做软件的微软公司。

Lisa 和 Mac 之间还有些细微的差异。比如关闭窗口需要点击的次数——Lisa 需要点击两次，而 Macintosh 只需一次。Lisa 里进入计算机的入口叫做 Filer，而 Macintosh 里的叫做 Finder；但区别并不仅限于名字的改变，使用者可以通过 Finder 启动程序，打开、关闭、命名、删除文件，还可以进行搜索。Lisa 的

程序通常都在桌面上运行，随开随用。但是 Macintosh 可没有奢侈到有这么多的资源供挥霍。在一台处理能力较低的计算机上创建 Finder 这个万能的潜望镜和向导可不是什么容易的工作。它不仅要能完成工作，还要方便使用，不容许系统崩溃——这可是个不小的挑战。

这一挑战落在了布鲁斯·霍恩身上。他是 20 世纪 70 年代阿兰·凯伊在施乐公司帕洛阿尔托研究中心所做用户实验中的少年之一。布鲁斯·霍恩竭尽全力工作，而计算机在 1984 年 1 月底上市的最终期限已经迫在眉睫了。于是，之前在帕洛阿尔托研究中心工作，后来加入苹果公司的史蒂夫·卡普斯被派来助布鲁斯·霍恩一臂之力。卡普斯回忆说："我只是来给项目研究增添动力，或许能提出些新想法，为设计出谋划策。"霍恩一直以来都在尝试利用 Macintosh 的视觉化理念来编写 Finder 程序，也就是说以图标为基础。有段时间，卡普斯和霍恩共同努力，研究出了基于图标的 Finder，但是这种图标的方案严重影响了程序运行的速度——只有 128 KB 随机存取存储器的计算机出现这种情况也是很正常的。（2001 年，廉价的台式计算机都有 64 MB 的 RAM，是 1984 年 Macintosh 的 500 倍。）

卡普斯决定试验一种带有一系列选择的目录式 Finder，每一条选择都呈带状显示在屏幕上，通过点击可以下拉显示。他回到家，花了一个晚上的时间，试验这种相对简约的视觉化的方案。计算机运行正常，卡普斯说："说不出原因，但给人感觉良好。"在 Macintosh 的团队中，"感觉良好"的测试至关重要。"但这么做也是因为有一些局限性，"卡普斯回忆说，"Mac 像一个神话，带有一种神奇的力量。但鉴于时间紧迫，技术上的可行性和我们当时的灵感，总要做出妥协。真正让我感到惊讶的是，这么多年过去了，我们现在的工作环境还来源于当时做出的决定。"

Macintosh 在 1984 年向世界展示的桌面设计理念之所以能够经久不衰、广

为流传，主要有两个原因。首先，不管库比蒂诺年轻的反叛者们是否意识到，苹果公司 20 世纪 80 年的用户界面其实是以 20 世纪 60 年代的研究成果为基础产生的。第二个原因是，他们真正关心普通人在使用计算机时遇到的困惑，并一般依此作出深思熟虑的选择。

思考过程本身就具有启发性。在用户界面领域，一个很重要的争论点就是鼠标上按键的个数，这一争论持续至今。在施乐公司帕洛阿尔托研究中心，Alto 有 3 个按键，正如当初道格·恩格尔巴特在斯坦福研究中心所设计的鼠标一样。但是在施乐之星的用户测试中，施乐公司注意到，上班族觉得 3 个按键会把人搞糊涂。因此公司推向市场的第一款鼠标就只有两个按键；微软公司也效仿施乐，使用只有两个按键的鼠标。

然而苹果公司却选择了一款单键鼠标，Lisa 和 Macintosh 都是如此。并不是所有人都喜欢单键鼠标，但是苹果公司在选择的背后有自己的用户哲学。特里布尔回想往事，对这一哲学作出了解释。加入苹果公司的时候，他还在华盛顿大学学习神经学。为了取得学位，他工作之余还要回到学校完成学习。即便现在，特里布尔也不仅仅是位软件工程师，他还是名有行医资格的医生，每年都参加培训和行医执照的年检。他想起在西雅图，路过一家电子游戏厅时驻足观看几个八九岁的孩子玩较为复杂的战争策略游戏，如太空侵略者和太空陨石歼灭战等。他们看有经验的玩家玩一两次就学会了，并没有查看操作指南。

这给特里布尔留下了深刻的印象。“关键就是你看别人这么做，自己也就学会了，”特里布尔说，“这就是苹果公司在 Lisa 上采用、并在 Mac 上发扬光大的方式。这最大程度地体现了简单易学——说用法简单恐怕不太贴切。就好像是在医学院教授一个新步骤——看一遍，自己做一遍，就可以去教别人了。这对视觉化的东西都适用，Mac 就是个视觉化的东西……这就是我们使用单键鼠标的原因。有一个按键时，你可以边看边学习；要是按键多了，你反倒看不清别人是如何点击的了。”

第 9 章

为每一个人编程：让用户自己动手

尽管众多软件开发人员一直在努力，但迄今为止，仍然没有任何编程语言能够帮助普通计算机用户跨越与计算机专业人士之间的鸿沟。多年来，从FORTRAN 和 COBOL 到 Visual Basic 和 Java，进步巨大，这使编程向更多的计算机业内人士敞开了大门，但却始终未能实现让普通用户自己编写程序的承诺。其他行业有一些值得借鉴的例子。电话服务刚刚扩展到社区的时候，一通长途电话至少需要经由两名接线员才能接通。全国范围内的电话服务所需要的人数更是无以计数。长途电话的普及看似毫无希望，但是随着信息交换技术的发展和其他科技的进步，出现了直拨电话技术。这使得原先由专业人员操作的劳动密集型工作全部自动化了。实际上，直拨把每个人都变成了接线员。

编程所面临的问题，无疑比 20 世纪初期的电话接线员困境要复杂得多，但是由用户 DIY 编程的想法却极具吸引力。长久以来，人们为普及计算机应用——用户界面设计的主要目的——做出了巨大的努力，而让用户为计算机自主编程，又是一大进步。这将有利于更加广泛地传播信息技术，赋予用户更多的主动权，并提高生产效率。

让每个人都成为程序员的想法是很难实现的，甚至可以说是一个幻想。不过，借助为特定任务开发、有着特殊目的的工具，如数据库编程、电子表格和网站建设，人们在这方面确实取得了一些进展。这些工具壮大了程序员队伍，供专业人士和非专业人士共同使用。这一成功源自编程视角的转换：从编写如何进行特定计算的方法，转换成了陈述用户的需求。

传统的编程语言都是程序化的，通常是通过描述程序来解决问题，告诉计算机“做这个，然后做这个，接着再做这个”，等等；而用来制作网页的非标准语言如 SQL 和 HTML，描述了用户希望进行的操作：从数据库中调出一类信息，或者给网站添加某种功能特性。像 SQL 和 HTML 这种混合语言被视为标示性语言。麻省理工学院两个年轻的研究生创造的现代电子数据表，就不是一种语言，只是采用了和 SQL 和 HTML 一样的标示方法，关注用户需求，然后通过软件翻译，将用户的要求传达给计算机。集用户的视角、灵巧的设计和计算机不知疲倦的即时重算功能于一身，电子数据表不仅仅是一个屏幕上的分类账页，也成了检验想法和其他可能性的模型工具。

探寻能够让用户自己编程的软件，早在 40 多年前就已经开始了。IBM 在 20 世纪 50 年代遇到过一个棘手的问题。当时公司推出了几款计算机样式，成为美国领先的计算机制造商。但是其市场过于高端，主要是政府机构、航天器制造商和其他能够负担得起昂贵机器的企业团体，而且计算机需要由程序员操作才能进行日常工作。IBM 当时还没能说服数以千计的其他商业用户购买其产

品。为了拓宽计算机用户市场，IBM 于 1959 年 10 月推出了 1401 计算机，租金为每月 2500 美元，以吸引广大的中端客户。1401 计算机的价格同几台 IBM 大型电子计算机的价格之和相差无几，并能做同样的工作。作为程序存储计算机，它通过运行程序员编写的程序能够完成各种各样的任务。然而，IBM 的商用计算机客户担心，如果购置一台这样的计算机，就要雇用程序员。当时，程序员给人们留下的印象是工作自由散漫、不守纪律——这是很多公司尽可能避免的一个管理难题。

因此，1959 年，IBM 指派了两名程序员芭芭拉·伍德和伯纳德·西克维兹，来完成开发软件解决方案的任务，目的是简化程序存储计算机的转换工作。[1]他们的解决方法是 RPG（Report Program Generator，报表程序生成器）。1961 年年初，在第一台 1401 计算机发货几个月后，报表程序生成器就问世了，并很快风靡一时。它以首字母 RPG 为大家所熟知。用户填写“说明表单”，叙述需要计算机完成的商业问题，例如工资表。用户自行限定自己想要计算机使用的输入数据、期望的输出形式和需要执行的计算。对用户来说，报表程序生成器的方式与 IBM 计算机熟悉的“打孔、分类、制表的口诀”极其相似，而且简化后的编程“语言”经过几天的培训即可上手。之后，上述简单说明便由生成器程序翻译成计算机语言。报表程序生成器在加法机和计算机之间搭建了一座软件桥梁。IBM 的官方历史学家在 1986 年写道，“在之后的几十年中，随着科技的进步，计算机会更加普及”，报表程序生成器的概念将“发挥更大的作用。”

唐·钱伯林在加利福尼亚的坎贝尔市长大。[2]坎贝尔市在发展成为赫赫有名的“硅谷”之前，只是圣何塞市附近一个以种植李子和杏树闻名的小镇。钱伯林生长在“史波尼克①时代”，当苏联的小型卫星发射升天时，他刚刚读八年

① Sputnik，1957 年苏联发射的第一颗人造卫星。此类冒险活动对世界各地人们的心智造成了影响，这些人在此影响下尝试决定自己未来的路。——译者注

级。当时全美国都在鼓励儿童对技术领域进行探索。钱伯林是中学校长的儿子，无需鼓励，就对科学很在行。他喜欢制作飞机、火车和火箭模型。在他家并不宽敞的车库里，钱伯林自己制作的、各式各样的模型就占了不少地方。钱伯林初次接触计算机是在哈维姆德学院，这是加利福尼亚一所小型的精英学院，专门教授数学、科学和工程学。他回忆说："在 20 世纪 60 年代，计算机还是地下室里的大型机器，但那时人们就意识到了计算机未来发展的潜力。"学院注册了半小时的 IBM 大型计算机使用时间，钱伯林的第一个程序是用 FORTRAN 编写的一字棋游戏，而他这么做只是为了好玩。

钱伯林成绩优异，在斯坦福大学读研究生时，获得过美国国家科学基金会的奖学金。取得博士学位之后，他于 1971 年加入 IBM 公司位于纽约郊外的华生实验室，从事研究工作。他的第一个项目是实验分时计算机操作系统，但是不到一年这个项目就以失败告终了。他的上司决定将小组的工作重心转移至数据库软件。20 世纪 70 年代初期，随着数据存储技术的进步，数据库技术在研究界蓬勃发展，促使大企业对有效使用计算机的需求更加迫切。在计算机早期发展的过程中，数据都按顺序存储在磁带上，每段磁带负责一项特定的工作，例如库存管理、采购或者应收账款等。数据限定在线性磁带上，因此，想要在多个应用程序上使用商业数据就十分困难。1956 年，IBM 开发了第一个磁盘式硬盘，改善了这一困境。硬盘支持数据在任何位置的"随机存取"。这样数据就不再受磁带上线性顺序的约束。它们能以各种方式组织到一起，数据和图片以树或者网络的形式分布在硬盘上。硬盘为数据库软件的创新开辟了新天地——程序能够存取、操控和组织信息。随着微处理器芯片的发展，磁盘驱动器不断得到改良。20 世纪 70 年代初期，这一发展潜力日益明显。储存在 6.45 cm^2 的磁盘上的数据量每隔 18 个月就翻一番，从而大幅降低了数据存取时间和成本——现在还保持着迅猛的发展势头。（1956 年，1MB 数据的磁盘存储器的价格为 1 万美元[3]；而 2001 年，配置 20 GB 也就是 200 亿字节数据容量硬盘的一台个人计算机还不到 1000 美元。）

20 世纪 70 年代初，钱伯林开始做研究的时候，数据库领域都是以查尔斯 · 巴赫曼的理论为主导的。巴赫曼是通用电气的软件设计人员，后来受聘于霍尼韦尔公司。1973 年，巴赫曼在他的论文“作为导航员的程序员”[4]中，把数据库技术描述成在存储的数据记录中使用软件指针作为向导的导航工具，就好像在森林中沿着一条路前进。后来，钱伯林开始研究这种复杂的、导航式的方式。“我就是喜欢它，觉得它很棒，”他回忆说，“一研究就是一整天。这是真正的谜题。”钱伯林和他在纽约约克镇高地的 IBM 实验室的同事隐约获知，公司位于圣何塞的实验室里有一位受过牛津大学教育的数学家特德 · 科德也在进行着类似的研究。钱伯林还记得，科德的工作成果给他的第一印象是，看起来有点古怪，像是“某种奇怪的数学概念。没有人把它当回事”。

但是 1972 年科德访问约克镇高地并讲解了自己的工作之后，钱伯林对他的研究开始真正重视起来。[5]科德将数据库解释成“关系”模型。简单地说，科德把数据看成是事物之间的联系——例如雇员、工资、办公地点，等等。相关联的数据可以组建成一个表。科德坚持认为，数据不应该被看成是一个需要导航的、类似于森林的网络。相反，他指出，数据应当存储在表中，并通过各式各样的方式相互连接。在这种相互关联的方法中，用户能够通过陈述式的方式，对数据提出查询或者请求，就好像一个人提出一个问题。据钱柏林回忆，在 1972 年的那次讲座中，科德提出了一个假设性的查询请求，“找出薪水高于经理的员工”。坐在观众席当中的钱柏林想象得出，如果他使用导航式的方式，这个查询请求“需要 5 页长的程序在指针和资料的迷宫里导航”。但是，他回忆说：“科德就好像只是随手一写那么简单。”对于钱柏林，这是一次“对话的经历”。直到这时，他才真正领悟了科德的观点所具备的力量。

实际上，科德的观点已经形成很长一段时间了。[6]他在 1970 年发表了一篇名为《大型共享数据库的关系数据模型》的论文。文中宣称网络导航模型存在严重的缺陷，会“大量地产生混淆”。而他的关系模型，他坚持认为“在很多方面都略胜一筹”。事实也确实如此，但是科德的观点并不仅仅是对数据库计

算中主导思想的攻击，同时也提出了一个带有神秘数学概念的理论。大多数计算机业内人士并不明白科德在说什么，但是钱柏林在 1972 年那天却豁然开朗，而且他对如何将理论转变成可以使用的数据库软件有了一个模糊的想法。IBM 的项目叫做 R 系统，而这个基本上由钱柏林创建的数据库编程语言就是 SQL 语言，也就是结构化查询语言。

1973 年，钱柏林和约克镇高地的其他一些人搬到圣何塞，研发 R 系统，也就是关系数据库的原型。虽然这是一项研究项目，但从本质上看，R 系统研发团队就像一支尖端的工程特遣部队。他们的任务就是将理论付诸实践，而不是单纯地空想。他们建立了一个工作系统，包括工业化的交易处理软件和为非专业人士打造的专门语言。科德并不是 R 系统的一分子，与其说他是项目领导人或经理，还不如说他是个幻想家。远在美国东海岸的研究负责人考虑让科德在 R 系统研发中扮演某种角色，但他们并不想让他运作该项目。因此，科德始终与该项目保持着一定距离，只是偶尔充当一下项目顾问，大多数时间，他都忙于在专业会议上向 IBM 客户宣传他的数据库理论，或者研究他感兴趣的课题。钱柏林这样比喻科德的角色："特德 · 科德提出了新房子的概念，然后，那些设计建造 R 系统的人据此设计出详细的蓝图，打好房子的地基，之后开始做木工活。特德是个理论家，主要进行培训和指导，而非系统建造者。天才的建筑师未必都是做事有效率的木匠。"

R 系统项目最终汇集了 20 名成员，主要由年轻的研究员组成，由经验丰富的项目经理领导。这是一个新的领域，因此团队中并没有熟知该领域尤其是关系数据库的"专家"。这些年轻研究员的背景五花八门，聚集于此的原因也各不相同。吉姆 · 格雷在约克镇高地工作[7]，研究应用计算机解决社会问题。例如，IBM 就为纽约的荷兰隧道做过运输流量的优化模型。有段时间，人们对计算机模拟有益于社会工作持乐观态度。麻省理工学院的杰伊 · 福雷斯特的理

论曾被罗马俱乐部[①]借鉴，作为《增长的极限》一文的理论基础。这篇反乌托邦的文章发表于 1972 年，认为如果没有大规模政府项目的支持，人类将很快面临能源紧缺和人口爆炸等问题。格雷曾专门开展过一个研究项目，力图证明福雷斯特模型的缺陷。

虽然工作本身极具吸引力，但是作为一个地地道道的加利福尼亚人，格雷厌恶生活在美国东部，他尤其讨厌这里阴暗寒冷的冬天。他对上司说自己不能再忍受纽约的冬天了，并且要求调到加利福尼亚工作。然而他被告知那边并没有职位空缺，因为在经济低迷时期 IBM 冻结了招聘计划。而且上司觉得格雷夸大了自己对积雪和雨水的厌恶之情。但是格雷并不是闹着玩的，他决定逃离这片纽约郊区。他回忆说："和这里比起来，加利福尼亚简直就是天堂，温暖的阳光、热情的人们，一年四季都有成熟的番茄。"格雷辞去了工作，跳上了他的大众甲壳虫汽车，一路穿越美国回到了加利福尼亚。当他出现在位于圣何塞的 IBM 实验室的门口时，那里正巧有个职位空缺。格雷加入了关系数据库研究小组，继续研究如何阐述交易处理软件的基本属性——这是一项至关重要的基础技术，它将信息从数据库中提取出来，并用于世界金融市场、航空公司机票预定系统和银行自动柜员机网络中。1999 年，格雷成为微软公司的研究员，并以出色的工作成果获得了业界享有声誉的图灵奖，而这一切都多亏了他当时坚持回到了加利福尼亚。

自从 R 系统项目启动以来，IBM 研发部门主管拉尔夫·戈莫里对该项目就极其热衷，给予大力支持。R 系统的工作正是戈莫里的热情所在——让计算机更加简单易用。数据库软件的研究越做越大。"这并不是毫无目的的，"格雷回忆说，"这项研究的重点在于对存取和组织信息的方式进行一次大革命。"

在科德的设计中，他提出了两种可能的关系数据库语言。这两种语言都具有数学特征，还包含许多希腊字母和注脚（类型线下面的数字和符号），只有

① 罗马俱乐部是关于未来学研究的国际性民间学术团体，也是一个研讨全球问题的全球智囊组织。——译者注

经过训练的数学家才能读懂，而且无法在键盘上打出来。钱柏林说："显然，如果完全听从科德的观点，只依靠这些符号，那么不可能在市场上大卖。"因此钱柏林和他的同事雷·博伊斯开始着手解决新数据库系统的用户界面问题。钱柏林和博伊斯早期曾在 SQL 语言上密切合作，但是项目刚开始一年，博伊斯就在 1974 年因为脑动脉瘤与世长辞了。他们想开发一种具有特殊用途、易于理解并能用计算机编译的语言。通过这种关系型的方法，人们能够以一种更加自然的方式向数据库提出问题，因为这与导航模式不同，问题本身并不包含通向数据的途径。

要想达到预期效果，这种语言就要经过精心设计和规划，达到一种平衡：有一定的构造使编译器能够翻译给计算机，并且像自然语言一样让人类觉得简单易用。钱柏林和博伊斯设计的语言类似于一种混杂英语。事实上，它起初被称为 SEQUEL（Structured English Query Language，结构化英语查询语言），后来 IBM 的律师发现 SEQUEL 是一家英国飞行器制造商的注册商标。钱柏林说这样也好，用英语做参考就有些误导了。他解释说："这不是英语，因为对于数据查询来说，英语简直糟透了，容易引起歧义。"

为了使 SQL 语言具备有规则的结构，能够被计算机处理，"动词"或者说数据操作指令被限制在了大约 8 个基本概念中，例如 SELECT、FROM、WHERE、GROUPBY 和 ORDERBY 等。被查询的数据也同样以一定的结构格式储存起来——如同特德·科德在他的文章中所描述的相关的数据表。这些表可以包括雇员（简称为 EMP），在一张大的表单里面有他们的姓名、部门、工作地点、工作职责和薪酬。因此使用 SQL 语言查询"查找（FIND）员工玛丽·史密斯的工资"显示如下：

```
SELECT      SALARY
FROM        EMP
```

```
WHERE           NAME="MARYSMITH"
```

再看另一个稍微复杂的例子，一家公司希望查看某些客户的信贷额度。客户数据储存在关系数据库中，其中包括他们每个人的姓名、居住的城市、州、邮政区号和信贷额度。因此用 SQL 语言查询“找到信贷额度高于 1 万美元的客户，并将他们按照从高到低的信贷额度的顺序排列”显示如下：

```
SELECT          NAME, CITY, STATE, ZIPCODE
FROM            CUSTOMER
WHERE           CREDITLIMIT>10000
ORDERBY         CREDITLIMIT          DESC
```

如今，世界上 90%的商业数据储存在关系数据库中，而钱柏林的 SQL 语言是占据主导地位的数据库语言。这是很多人共同努力的成果，其中包括帕特·塞林格，她编写了“优化器”（optimizer），能将 SQL 语言翻译成详细的计划，用来重新找回数据。而吉姆·格雷发明了交易处理技术，使得 SQL 语言在现代电子商务中广泛应用。钱柏林将这一切融为一体，他知道怎样将这些碎片组合起来——数据定义、数据操作和数据表等。他看到了特德·科德观点的重大价值，并将其付诸实践。阿兰·凯伊曾经说：“观点的价值是 80 个智商分。”[8]钱柏林将一种精炼的、简化的视角带到了关系数据库软件领域。“钱柏林绝对是个优秀的程序员，他知道怎样将这些想法变成现实，”吉姆·格雷评论说，“所有这些对钱柏林来说是显而易见的，直到他把这些展示给我们看的时候，我们才明白过来。”

IBM 并没有很快地将 SQL 语言推向市场。1979 年，位于硅谷的一家新成立的公司——关系软件有限公司（Relation Software, Inc.），推出了第一个 SQL 语言数据库产品，这家公司在三年后更名为甲骨文公司（Oracle Corporation），其中一部分原因就是 IBM 实验室发表了他们自己的研究成果。关系数据库的研究也取得了其他的一些成果，例如伯克利的迈克尔·斯通布雷克和尤金·王

负责的政府资助的 Ingres 项目也发表了研究成果。但关系数据库思想的发展一直悬而未决，直到企业家劳伦斯·J. 埃里森将其价值挖掘出来。劳伦斯·J. 埃里森是甲骨文公司的创建者和首席执行官，他富可敌国。（埃里森很早就显示出了富有想象力的市场观念，他称自己首个产品为甲骨文第二版。但那实际上却是第一版，因为懂行的用户都尽量避免购买软件的第一版，认为里面的缺陷很多。）

1978 年的夏天，埃里森给钱柏林打电话，告诉钱柏林他发现一篇发表于 1974 年的文章是多么的有用。[9]这篇论文就是钱柏林和博伊斯写的《SEQUEL：结构化英语查询语言》。“我们写了篇文章，而拉里·埃里森以此建立了一家公司。”钱柏林指出。埃里森致电的目的不仅仅是为了交际问候，他很想从钱柏林那里得到详细的技术信息——也就是所谓的“误码值”——这样他的产品就可以与 IBM 的产品完全兼容。IBM 认为，即便对方是家新成立的公司，这在一定程度上也是在帮助竞争对手，因此他们拒绝了埃里森。IBM 的管理层不愿意采用 SQL 语言关系数据库技术，因为在 20 世纪 70 年代后期，公司的 IMS 导航产品销售火爆。而来自甲骨文和其他公司的外部竞争者，让 IBM 公司对来自于自己实验室的、具有非凡意义的数据库研究成果刮目相看，这实在是太讽刺了。

由于使用了人们熟悉的术语，关注人类的提问形式，而非徘徊在计算机底层的数位，SQL 语言极大地简化了建立数据库应用程序的过程。人们每天都在不知不觉中使用 SQL 系统。当商场里的机器读取信用卡时，或者人们在 ATM 机上按下数字时，这些数字都被 SQL 语言的语句用于存取数据库，更新用户的信用卡和银行账户信息。尽管 SQL 语言取得了这样的成功，它还是与其原本的目标有一点距离。当初，钱柏林和他的同事以为普通大众能够通过自行使用 SQL 语言，以一种全新的方式与数据互动。但是从很大程度上讲，事实并非如此。使用 ATM 机的用户确实直接进入了巨型数据库，但藏在按键或触摸屏背后的却是 SQL 语言。

1975 年，钱柏林终于意识到将 SQL 语言普及给除专业计算机程序员以外的普通大众并非易事。R 系统项目招募了一批在圣何塞州立大学就读的大学生，来试验这些随机选择的学生们对于 SQL 语言的反应。结果并不令人满意。经过努力地学习，他们能够掌握 SQL 语言，但是这个过程耗时太长，而且学生们在使用中也错误百出。“我们在 1975 年的时候对于普通民众使用计算机语言过于乐观了。”一天，坐在自己 IBM 办公室里，俯瞰圣何塞郊外褐色山丘，钱柏林评指出：“计算机按照字面意思，逐字逐句地接受每一条指令，它们并没有常识。因此即使是像 SQL 这样含有英文单词的语言，也无法像自然语言一样通俗而灵活。”

1970 年，丹 · 布莱克林和鲍勃 · 弗兰克斯顿在麻省理工学院相遇。布莱克林来自于费城而弗兰克斯顿来自纽约，但他们的背景却惊人地相似。这两个人合作开发的产品在 1979 年引发了一场个人计算机的革命。他们的友谊一直持续到现在，两人都住在波士顿郊外，两家之间车程不远。当被问道为什么两人在从一开始就这么亲密无间时，弗兰克斯顿坐在他位于堪布里奇的家中办公室——一个塞满了机器、配件和零部件的个人计算机实验室——回答说：“很难说这是为什么，但我们就是合得来。”

他们两人都来自于勤劳努力的犹太人家庭。这样的家庭鼓励追求知识，同时也不乏创业的才干。布莱克林和弗兰克斯顿都对计算机显示出了极大的兴趣。在 20 世纪 60 年代，他们还是少年的时候，就想方设法地接触到了大型公共机构计算机。布莱克林的父亲巴鲁克在费城经营着家族的印刷生意，但其实他的专业是机械工程，毕业于宾夕法尼亚大学。[10]“父亲曾经让我坐在他的腿上，手把手教我使用计算尺，”布莱克林回忆说，“家里到处都充满了工程学的氛围。”孩提时，布莱克林很快就玩腻了建筑拼装玩具而爱上了电子产品，他使用希斯套件自己做了个短波收音机，这台立体声收音机在他家用了很多年。

他还制作了一个厨房对讲机，这样他母亲鲁思就可以通过对讲机呼叫在楼上卧室中的布莱克林下来吃饭。

15 岁的时候，布莱克林在高中的分时系统终端上编写了他的第一个计算机程序。之前他都是编写一些将数字制成图表和将文本排序等简单程序。1967 年，布莱克林获得了一个名额，参加宾夕法尼亚大学摩尔工程学院开办的暑期项目。该项目让高中的理科学生能够接触计算机，他们的项目活动地点就在存放原始 ENIAC 计算机的房间下面的大厅里。随后布莱克林就在宾夕法尼亚大学沃顿商学院里大展身手，他开发了一个能够自动给标准型测验打分的程序。他的一款被称为 WhartFor 的软件，扩展了 FORTRAN 的一些特定功能，后来还赢得了为高中生设立的国家科学奖。1969 年秋天，布莱克林进入了麻省理工学院，他本打算学习数学专业，但后来很快就放弃了数学，改学计算机专业。

弗兰克斯顿的父亲本杰明经营一家为电视机维修厂商提供电子零件的制造厂。[11]他的母亲桃乐茜在亨特学院讲授地质学和地理学。在他母亲的坚持下，家族的姓氏从弗兰肯斯坦改成了弗兰克斯顿。虽然其真正的发音为弗兰肯斯泰因，但无论如何这个名字都让人联想到玛丽·雪莱①那部著名的哥特式恐怖小说和它的各种电影版本。“你可以想象得出我的叔叔鲍里斯向人推销人寿保险时的情境。”弗兰克斯顿笑着摇摇头说。13 岁的时候，他通过母亲工作的亨特学院开办的夜间班开始渐渐学习如何使用计算机。这个夜间班是为学校员工开办的，但是似乎没有人反对一个少年加入其中。而且在亨特学院的职工中也没有一个人能比这个坐在教室后排的瘦弱男孩更好地运用指令了。他用 FORTRAN 编写的第一个程序是将几十年中的闰年都挑选并打印出来。

斯泰弗森特高中是纽约市一所公立学校，入学条件极为严格。弗兰克斯顿是个聪明的学生，但是他从不专心，对待课业也没有耐心。在斯泰弗森特高中就读有一个优势，那就是学生们可以接触到附近纽约大学的 IBM 大型计算机。

---

① 玛丽·雪莱，英国著名小说家，因其创作的文学史上第一部科幻小说《弗兰肯斯坦》，而被誉为科幻小说之母。她是英国著名浪漫主义诗人雪莱的妻子。——译者注

在编程的过程中，弗兰克斯顿说他找到一项"正巧与他产生共鸣"的活动。他很享受编程所需的精湛技术，享受建造起某样东西的宏伟目标，也享受与机器互动时的触感。"这就像是毒品。"他说。随着他水平的提高，弗兰克斯顿发现他的技能有着广阔的市场前景在等待着他。他找了做程序员的暑期工作和兼职工作。在大学的最后一年，他还在斯泰弗森特高中帮助讲授编程课程——"这是我在高中唯一得过满分的课程"。其中的一个兼职工作是在华尔街的投资银行怀特威尔德公司（WhiteWeld&Company），开发用于分析股票和债券市场趋势的数据库和计算机软件。即使后来到了麻省理工学院，弗兰克斯顿还在做这份怀特威尔德公司的工作，偶尔往返于学校与纽约之间，或者通过他公寓内的电传打字机工作，这使得他一年的电话费就高达 1500 美元。"你一定要做能让你学到东西的工作。"他解释说。

布莱克林和弗兰克斯顿于 1970 年相遇，他们同是麻省理工学院 MAC 项目的程序员，这个项目着手研究野心勃勃的 Multics 分时系统。Multics 项目——由麻省理工学院、通用电气和 AT&T 公司的贝尔实验室共同完成——雇用了大约 100 个软件开发人员，其中只有少数几个是像布莱克林和弗兰克斯顿这样的学生。他们通常都是晚上过来然后工作一整夜。布莱克林和弗兰克斯顿很快就成为好友，经常在黎明的时候由弗兰克斯顿驾车一同去吃早餐。那时，他们甚至还相约有一天要一起开办公司。"他和我都是创业型的。"布莱克林回忆说。

但是差不多过了 9 年，布莱克林和弗兰克斯顿才建立了软件艺术公司（Software Arts），专营一种名叫 VisiCalc 的电子数据表程序。在麻省理工学院，布莱克林为 Multics 编写简单的计算程序、编译器和解释程序。毕业之后，他在数字设备公司的排版组工作。布莱克林的工作是要前往美国和加拿大的各个报社，帮助将原先的手工排版印刷转变成为计算机排版和屏幕编辑系统。他还清楚地记得 1974 年在《堪萨斯城星报》昼夜不停地工作，为尼克松水门事件录音带的电子版排查错误的情景。在数字设备公司，他要为那些不喜欢计算机的人们——这是一个远离麻省理工学院实验室和 Multics 项目的世界——设计

文字处理和编辑软件。他回忆说“这是用户界面设计的教育”，因为“报社里的人们是真正的用户”。1976 年，布莱克林接受了一份给电子收银机生产商编程的工作，目的是为了给自己来年上哈佛大学筹措学费。但是这份工作也给了布莱克林一个与摩托罗拉 6800 微处理器近距离接触的机会。摩托罗拉 6800 微处理器是当今苹果公司个人计算机微芯片的前身。

比布莱克林年长两岁的弗兰克斯顿直到 1976 年都一直在麻省理工学院从事研究工作。在那里，弗兰克斯顿是学生信息处理委员会的创始人，这是一个分时项目，目的是让大学生们接触到计算机。他的硕士研究项目关注于如何能够让计算机网络成为提供服务的市场——这是电子商务和在线交易的雏形。一直到 1979 年，他都在给包括怀特威尔德公司在内的一些公司做合同工，为他们提供编程服务。

大体来说，VisiCalc 的故事版本通常是：一个哈佛商学院的学生与他一个计算机外行的好朋友发明了电子数据表，个人计算机行业中第一个“杀手级应用程序”。这个版本在某种程度上是真的。但是 VisiCalc 是由两个不满 30 岁的、在计算机行业有各式丰富经历——数据库、金融程序、编译器、解释程序、文本编辑和用户界面设计等——的年轻人共同创造的。“我们所有的计算机科学方面的背景融合起来就有了 VisiCalc，”弗兰克斯顿指出，“这就是我们两个家伙创造的从无到有的神话。”

1977 年的秋天，布莱克林进入哈佛大学商学院学习。翌年春天，布莱克林坐在奥尔德里奇大厦 108 室，思索着怎样才能更好进行金融计算。按每月、每季、每年来计算公司的销售和利润，再加上其他一些变量例如市场、制造、运输和利率的变化，基本规律都是一样的。这需要写在一张方格纸上分开计算，根据要求加上、减去或者相乘。如果假设发生了改变，就要重新计算，每一次新的计算都有可能出现错误。而这样的工作就是几代商学院学生、经理和金融

分析师们的宿命。德州仪器公司和惠普公司制造的手持计算器使得计算工作不像过去那样令人生厌，但是布莱克林觉得肯定还有更加聪明的方式来进行这样的计算。说到底，计算机就是用来做大量计算的。

在哈佛大学有一件令人愉快的事情，那就是学校中使用 BASIC 编程语言的分时系统是免费的。布莱克林试验过，但是运行 BASIC 语言时分时系统有延迟，因此远远不能满足他所期望的快速计算和重算的要求。布莱克林在麻省理工学院见过施乐公司帕罗奥图研究中心的 Alto 计算机，十分欣赏个人计算机快速的互动反应时间和图形式文字编辑器 Bravo 的视觉效果。但是施乐公司位于帕罗奥图的实验室从来都没有试验过电子数据表。“我们在帕罗奥图研究中心没有会计。”[12]巴特勒·兰普森解释说。

布莱克林也许没有做过会计，但是他和弗兰克斯顿曾经学习过会计学，而且他们与施乐公司帕罗奥图研究中心大多数研究员不一样，他们俩都对商业很感兴趣。因此布莱克林向设计电子数据表发起了挑战。他仔细考虑了数字和文本怎样在屏幕上呈现，怎样进行计算，怎样处理格式化和数据结构才能使用户理解，同时合理地翻译给计算机。布莱克林决定，不仅要将纸版表格自动化，他希望有更多的灵活性——能够呈现给用户不同的视角——通过编程的魔法。“我觉得你应该能够转变看法。”他说。

布莱克林决定先同免费的顾问——他在哈佛大学的教授们讨论一下他的想法。在那个时候，他还只是有个想法。起初他所询问的几个教授倒是很支持他，但是他的金融学教授告诉他市面上已经有许多种 FORTRAN 的金融程序了。他告诉布莱克林说可以和一个大学二年级的商学院学生丹·费尔斯塔拉谈谈，这个学生成立了个人软件公司，准备涉足计算机商业领域。布莱克林还记得他的教授说，费尔斯塔拉能够让布莱克林知道的自己想法被误导了。布莱克林给费尔斯塔拉打电话，只和他进行了简短的交谈，并没有告诉他自己正在做什么。

整个 1978 年的夏天，布莱克林都一直在思考他的电子数据表，并且自己

做了一些调查。诚然，市场上是有金融预测程序，但它们主要都是针对大公司，在分时系统上运行每月需要花费数千美元。到了秋天，布莱克林又与费尔斯塔拉取得联系，与他更详细地讨论了自己的想法。费尔斯塔拉借给布莱克林一台Apple II计算机，与他一同编写“周末原型”。这是个粗糙的仿制品，有行和列，只能进行简单的算术运算，仅此而已。费尔斯塔拉想向市场推出一款支票簿程序，因此他鼓励布莱克林继续他的实验。布莱克林打电话给他的老朋友鲍勃·弗兰克斯顿，询问他是否愿意与自己一起从商，就像原先他们俩曾经讨论的那样。很快，他们就定出了计划和分工。布莱克林继续留在哈佛大学，负责设计工作，而弗兰克斯顿编写程序。“弗兰克斯顿离开后就动手了。”布莱克林回忆说。

他们的工作方式被弗兰克斯顿称为“快速迭代原型”（fast-iteration prototyping）——两个人密切合作，通过不断的改进和实验使项目稳步向前发展。一个至关重要的设计决定是使用“格子”，类似于会计和工程师们使用的纸版的格子样式。“你要给人们一个他们能理解的比喻。”布莱克林说。格子的概念并不只是在外观上让人们觉得熟悉，对于计算机来说，格子就是一个数据结构，格子的每一个单元都是数据的一个计算机位置。用户可以改变数字，将它们按照不同的顺序，移至不同的单元，然后计算机可以让这些数字按照列、行或者任意选定方式即时计算出来。它将线性金融预测程序所不具备的灵活性赋予了 VisiCalc。它是一个可视的工具——从它的产品名称中也能体现出来，“可视计算机”（VisibleCalculator）的缩写——同时也是个强大的工具，这很大程度上是借助了个人计算机即时重算的能力。VisiCalc 是询问各种各样假设分析问题的手段。“你不用事先做程序规划，”弗兰克斯顿解释说，“你就好像在制作黏土模型。它是能够检验你的想法的工具。”

VisiCalc 在当时是个紧凑的、运行快速的程序。用户可以很流畅地上下或

者左右查看。光标的运动和键盘输入都能够即时地显示在屏幕上，给人一种高速运行的互动体验。考虑到 Apple II 被设计成为游戏计算机，弗兰克斯顿在为 Apple II 编写 VisiCalc 的程序时，没有加入任何冗余的功能。虽然按照现代的标准来看，VisiCalc 是个很小的程序，但 VisiCalc 充分利用了 Apple II 能够提供的每一点内存和处理能力。VisiCalc 占据了整台机器。Apple II 没有操作系统，因此弗兰克斯顿使用汇编语言为 VisiCalc 的操作（直到最细微的操作）编写程序。例如，如果有人打字速度太快，弗兰克斯顿就要写入的指令“轮询”键盘敲击——暂时存储字符并马上将它呈现在屏幕上，以保证没有漏掉任何键盘输入或者出现恼人的长时间延迟现象。

VisiCalc 以紧凑的方式将设计、细节和架构融为一体，技艺精湛。除了业界内一些精明的老手，不是每个人都能意识到 VisiCalc 的巨大成就。经验最为老道的是投资银行摩根士丹利公司的分析师本杰明 · 罗森[13]。罗森后来成为一名风险投资家，并成为莲花发展公司和康柏电脑公司的主要支持者，他曾在康柏做过几年的董事会主席。在 VisiCalc 上市之前几个月，罗森就看到了这个电子数据表。

在 1979 年 7 月 11 日一份呈现给投资银行客户的报告中，罗森写道：“在大型机和小型计算机当中，硬件的发展都远远超过了软件的发展速度。这一模式在新生的个人计算机领域中也是如此。今天，唯一对软件技术发展水平感到满意的个人计算机用户只是那些计算机爱好者们。他们自己编写程序……虽然很难用语言形容，但 VisiCalc 的视觉效果逼真。在几分钟之内，从来没有使用过计算机的人就能够书写和使用这个程序了。即使你用很浅显的英文进行操作，程序也能够用计算机语言来执行。但是对你来说，整个过程都是通过软件透明呈现的。你只需简单地在这个电子黑板上写下你希望它做的事——然后软件就替你完成了。”

“VisiCalc，”罗森总结说，“总有一天将会在个人计算机界独领风骚。”VisiCalc 确实扮演了这样一个角色。它证明了个人计算机不仅仅是计算机爱好

者们狂热追逐的产品，而且同样也可用作商业工具或用作其他用途。许多人购买 Apple II 就是为了使用 VisiCalc。随着软件艺术公司的成长，它陷入了困境。IBM 公司的个人计算机迅速在商业市场中占据主导地位，而软件艺术公司为其开发 VisiCalc 的速度却跟不上它的脚步。布莱克林和弗兰克斯顿与他们的发行商兼市场推广丹·费尔斯塔拉大吵了一架，并闹上了法庭。最后软件艺术公司被出售给了莲花公司，而莲花公司开发了 IBM 个人计算机使用的电子数据表软件 Lotus 1-2-3。但之后莲花公司又被微软公司及其 Excel 电子数据表所取代。

虽然布莱克林和弗兰克斯顿并没有因他们的发明而致富，但 VisiCalc 却是个教科书般的软件。它指明了方向，而随后出现的软件都跟随它的脚步。从编程的角度讲，VisiCalc 使报表程序生成器和 SQL 等声明式编程语言又前进了一步，至少在金融建模领域，普通大众可以自己编程。他们无需写下运算法则说明怎样操作；他们只需描述所希望的结果，其他事项由 VisiCalc 来完成。这种方式称为“实例编程”。在 1995 年的一次研讨会上，研究 SQL 语言的小组重新聚首，一位 IBM 的前研究员提到，20 世纪 70 年代末期，在约克镇高地实验室有一群人通过实例来编写程序。“你只需要告诉这个奇妙的软件你想要什么结果，它就能知道怎样得到这一结果，”IBM 的前研究员迈克·布拉斯根解释说，“但是，最后不了了之。实际上，你知道实例编程是什么吗？而它真的发生了。这就是 VisiCalc。”[14]

“如果你说 HTML 编程，每个真正的计算机工程师都会窃笑。没有人把 HTML 当做编程。”[15]布莱恩·贝伦多夫指出。布莱恩·贝伦多夫是协助完成阿帕奇项目的负责人，他的软件大受欢迎，为网络注入了动力。他说：“现在之所以有数十亿的网页，是因为用 HTML 来制作网页很简单。”

超文本链接标示语言是万维网的一种混合语，是互联网上文档呈现和交换的一种基本语言。HTML 与同为首字母缩写的 URL（统一资源定位符）和 HTTP

（超文本传输协议）是网络的三大支柱——这三个看似简单却功能强大的软件协议支撑起了一个新的信息与商务的全球媒介。网络的重要影响如同近 25 年来计算机界每一个重大发展一样。与大多数计算机的发展相同，它也是首先在美国受到了热烈追捧和商业开发，但是网络却不是来源于世界领先的美国大学或企业的计算机科学家，而是一名来自阿尔卑斯山脚下、日内瓦郊外的欧洲粒子物理研究所（CERN）物理实验室的英国物理学家的创造。

早在蒂姆·伯纳斯–李在欧洲粒子物理研究所创造了万维网的十多年之前，他就已经接受了解决难题的良好训练。[16]他的父母都是英国的数学家，他甚至还有基因优势。父亲康威·伯纳斯–李和母亲玛丽·李本身就是计算机界的先驱，他们都是为 1951 年的世界上第一台商用程序存储计算机 Ferranti Mark I 编程的人员。伯纳斯–李回忆起小时候，他的父亲为巴兹尔·德·费伦蒂所作的一次演讲。巴兹尔·德·费伦蒂以自己的名字成立了一家电子公司并担任主席。作为准备工作，伯纳斯–李的父亲阅读了大量关于人脑的资料，寻找计算机——没有想象力的数字处理器——如何才能变得像人类一样利用某种直觉联系。这一问题一直伴随着他，萦绕在这个年轻人的心上。

1976 年，伯纳斯–李毕业于牛津大学皇后学院。虽然他的专业是物理学，但他却具备工程师的天分。出于对计算机的痴迷和微处理器的价格降到大众可接受范围，伯纳斯–李采用摩托罗拉 6800 微处理器、一个临时组装的母版和一台用作显示器的旧电视，制造出了一台自己的个人计算机。后来，伯纳斯–李继续他的软件工程和设计，他早期在大型公司工作，同时也做些咨询和合同工作。他的第一份工作是在一家英国大型的电信设备供应商普莱思电信公司，研究信息传递和条形码技术。

之后，伯纳斯–李在一家位于多塞特的制造商 D. G. Nash 公司工作，主要负责为打印机编写排版软件。后来，他做了一年半的独立承包人，其中在 1980 年有 6 个月的时间是为欧洲粒子物理研究所工作，在那里他自己还制作了一个他称为“询问”（Enquire）的储存个人信息的程序。“询问”带有一些链接的特

征，是他后期成果的概念雏形。伯纳斯–李后来又在一个朋友刚刚成立的公司工作，将微处理器技术应用在便宜的点阵打印机上，使之能够打印出生动的图像。在这个项目上，他为打印机处理的文档编写了一个标示语言。

到了 1984 年，当伯纳斯–李再回到欧洲粒子物理研究所的时候，他已经有了 8 年的计算机工作经验，这些经验各种各样，看似没有太大联系。但他的大部分时间都用来处理与文档相关的实际问题。在这几年中，他一直都在思考一个很大的命题，如何利用计算机更加有效地组织信息。“回首过去，”伯纳斯–李回忆说，“我之前的这些工作经验使我学到设计万维网所需的各项技能。”回到欧洲粒子物理研究所之后，他开始着手软件和信息管理的课题。这时，他遇到了互联网产生之前庞大的体制环境：数千名研究员使用来自不同供应商的计算机系统工作，每一个系统都像是一个单独的信息孤岛。这种糟糕透顶的低效率大大地触动了伯纳斯–李，而且他说，这种问题由于“物理学家们自己编写软件的恼人习惯”而更加严重了。我们所需要的，伯纳斯–李确信，是一个能够存储并连接信息的网络版的“询问”程序——使工作更加有效率和合作性的工具。

在接下来的几年中，欧洲粒子物理研究所的实验室成了伯纳斯–李创造万维网的孵化器。他本来只是想为研究所创造一个更好的工作环境，但是他设计的这个解决方案最终却成了一个供很多人使用的、通用的解决方案。物理实验室的没有一个人像他这样真正对计算机科学产生兴趣。伯纳斯–李的项目最直接的目的是要在欧洲粒子物理研究所里更加有效地使用数据库和文档，但是他一直有着更长远的设想。物理研究所为伯纳斯–李进一步探究超文本和互联网领域——伯纳斯–李想将这二者和软件连接起来——提供了宽松的学术氛围和充裕的研究资金。

大量知识自动存储的概念至少要追溯到 1945 年的《大西洋月刊》上范内瓦·布什的文章“As We May Think”（成若所思）。在这篇文章中，他提到了

称为 memex 的能够存储数百万图书、报纸、杂志和其他出版物的个人信息机器。20 世纪 60 年代，道格拉斯·恩格尔巴特在斯坦福研究院致力于普及在线信息，用于“增强人类智力”。泰德·尼尔森这个打破陈规的福音传播者提出的权力归于人民的“计算机解放”大大地影响了个人计算机界，在 20 世纪 60 年代中期杜撰了“超文本”这个词——一个在存储于计算机中的信息世界导航的系统。20 世纪 80 年代后期，随着伯纳斯–李对其项目的进一步研究，他也接触到了本领域内的“领先技术”。一位在美国工作过的欧洲同事曾亲眼见识过美国所有的大学和研究实验室是如何通过互联网连接起来的，他极力鼓动欧洲粒子物理研究所也采用该技术。那时，欧洲并没有多少人对互联网感兴趣，但是伯纳斯–李却对此印象深刻。互联网民主化的连接能力使他认识到，这对他信息分享网络的想法很有帮助。它可以成为“全球超文本”，它可以成为万维网。

伯纳斯–李拜访了超文本和电子书籍的社团，希望能劝说他们使其产品能在互联网上应用。这样的话，伯纳斯–李说，他就可以不用自己建造，而是简单地购买他所需要的工具。这并非是一个不切实际的期望。举例来说，苹果计算机公司在 1987 年推出了 HyperCard，一个简单易用的、通过超文本连接的信息卡系统。用户可以在卡上创建信息——文档、地址、日程表和时间表等——然后通过点击从一张卡上跳到另一张卡上。但是伯纳斯–李曾经试图说服的那些超文本和电子书籍供应商们却没有意识到互联网的巨大潜能。当时，互联网的应用只是局限在大学和研究所等相对较小的群体当中，采用 Unix 语言，奉行极客文化。包括伯纳斯–李在内，没有人能够想到万维网会蓬勃发展，通过首先由网景公司推出的点击式浏览软件，万维网导航变得如此简单，更没有人能预测在 20 世纪 90 年代中期互联网大爆炸的到来。

因此伯纳斯–李只得靠自己继续前行。到了 20 世纪 90 年代末期，他已经开发出了 URL、HTTP 和 HTML 的原型版本——多媒体文档选址、连接和传送所必需的基本协议。他所使用的硬件和软件都来自于 NeXT 公司。NeXT 公司是一家工作站计算机制造商，由史蒂夫·乔布斯于 1985 年在被逐出苹果公司

之后建立。即使在多年以后，伯纳斯–李也对 NeXT 公司的 Unix 操作系统、用户界面——“美观、流畅、可靠”——和创建超文本程序的软件赞赏有加。1991 年，就在伯纳斯–李在互联网上发布自己的软件的时候，他拜访了史蒂夫·乔布斯和盖·特里布尔。盖·特里布尔离开苹果公司后加入了乔布斯的 NeXT 公司，出任首席软件工程师。但是 NeXT 公司的团队也没有意识到伯纳斯–李成果的重要性。“如果万维网还不存在，真的很难展示它，”特里布尔回忆说，“当时我们在 NeXT 公司，个个都在挠头。在他展示完毕后，我们都窃窃私语说‘唔，我们没明白。’我们就像当时的其他人一样，错过了它。”[17]

---

3 年后，就在万维网成为信息与商务的主流媒介之时，伯纳斯–李来到了麻省理工学院的计算机科学实验室，成为万维网联盟的董事。这是个为万维网设立标准的组织。伯纳斯–李的使命就是要让万维网成为全球性的信息空间，向任何人免费开放，而不是将其分割成独立的商业阵营。伯纳斯–李身材瘦长，细细的金色头发，不修边幅。他是个古怪的、有点专制的权威人士，实用主义和理想主义兼具。他并不是一个苦行僧式的人物。在 20 世纪 90 年代初期，他说如何利用他的发明来赚钱的想法“几乎每天都在脑海中萦绕”。但最后，他认为独立地工作，开放万维网，远离任何一家公司的影响似乎才是正确的选择。

伯纳斯–李的语速和他的思维一样快——有时甚至更快。“我说到哪里了？”他会在句中停顿下来询问。在欧洲粒子物理研究所里，他那些通晓几国语言的同事最后都要求他说法语，希望这样能让他的语速慢些。他有着英国人特有的自嘲式的幽默。在面对 1000 多听众的万维网会议上，他遇到了在展示某些软件时不可避免的计算机故障。他一边敲击键盘解决问题，一边像是旁白一样说：“唔，你们大概都在说要是换了你们自己是不是能把这东西弄好。”

对于伯纳斯–李来说，互联网是一个包含了他所呼吁的“宽容与分权管理”社会价值的世界。“为什么审查制度是错误的？”他反问道，“因为它通过中央

集权宣扬什么是对的。”他的这一解释体现了理性的工程学视角，而非主观的道德判断。他说，互联网的价值体系转化成了“简单和模块性”的软件设计原则。

在设计万维网的时候，伯纳斯–李就以这些价值和设计原则为导向。统一资源定位符或称 URL，最初被称为统一资源标识符（URI），它能够“在信息领域内指向任何文档（或任何其他类型的资源）”——一种像人们尽可能想象的那样宽容与分权的软件协议。超文本传输协议或称 HTTP 是一种从一台数据服务器计算机向无数台式计算机或手持装置发送万维网网页的通信协议。超文本链接标示语言是一种简单的超文本呈现方式，因此它每天都在被分解，在互联网中发送，再在数百万的台式计算机屏幕上，以多媒体万维网网页的形式重新组合起来。

计算机科学家们在最初看到伯纳斯–李的发明的时候，认为它不只是简单，简直就是愚蠢。他们说，没有一个真正的计算机系统工程师会设计一个像万维网一样的东西，每一次点击都会打开一个新的互联网传输协议连接。一些系统研究员们认为万维网混乱无序，并预言它没有能力处理日益增长的使用量，但是万维网却随之按比例增长，它大概是历史上做的最出色的软件了。伯纳斯–李在 HTML 语言上做了一系列精明的决定。他使用现今存在的标示语言家族中的标准句法——标准通用标记语言（standard generalized mark-up language），或称 SGML，因为专业人士对它再熟悉不过了。使用 SGML 使得 HTML 在超文本环境下更容易为人所接受，这在刚开始时是很重要的。SGML 将尖括号内的格式化命令赋予 HTML，作为指令的标记，称为“标签”（tag）。所以，<title>是开始一个标题（title）的标签，<body>表示开始一段文本的主体，<b>表示开始粗体文本，<u>表示下划线，<img src>表示插入图片。最初版本的 HTML 语言只包含这样的基础指令，而随后的版本又加入了一些自动的功能，例如包含一整套页面设计的式样表。

跟报表程序生成器（RPG）、SQL 语言以及 VisiCalc 一样，HTML 是一种

简化编程声明样式的工具——它描述出了问题和网页本身，而非从程序上和更贴近计算机的角度说明如何用传统方式编程。对于伯纳斯–李来说，他从一开始就不认为 HTML 编程是非专业人士能够胜任的工作，但在万维网普及之后，他并没有阻止他们这样做。在他与马克·菲谢蒂合著的《编织互联网》一书中，伯纳斯–李写道："我从来没有想让用户们看到 HTML 的源代码（用尖括号的东西）……但是 HTML 的易读性所带来的福音却在我们意料之外。令我感到惊讶的是，人们很快就能熟悉那些标签，并且直接开始编写他们自己的 HTML 文档了。"

第 10 章

Java：杂乱中诞生的新语言

1995 年 2 月某天的下午，约翰 · 盖奇的脑海里突然冒出几个简单的想法。于是他往办公室里探探头，向詹姆斯 · 高斯林索要了计算机电缆等物。詹姆斯 · 高斯林翻箱倒柜找到了盖奇想要的东西，但是他忍不住好奇：盖奇找这些设备做什么呢？盖奇解释说，他明天早上要在一个大会上做演讲，他想以蓬勃发展的高科技来闪亮结尾。事实上，盖奇准备展示的软件项目是高斯林在太阳微系统公司工作时一直跟进的一个项目。此前，这个项目从未公开过。虽然高斯林认为这个软件已经成型，但是，让盖奇一个人在参会的 500 个人面前应付它，他还是有些不放心。这次演讲顺利与否将取决于互联网连接是否顺畅。

高斯林回忆道:“处处都有可能失败，所以我只好同盖奇一起坐上了车。”[1]

他们翻山越岭，前往北接太阳微系统公司硅谷办公室的蒙特利参加会议。盖奇没有带换洗衣物和洗漱用具，只是打电话告诉妻子自己这几天不能回家了。

约翰·盖奇属于 20 世纪 60 年代的通才。[2]他是全美游泳运动员，因为沉迷于自由言论运动而放弃了加州大学伯克利分校的数学学位；他曾制作针对尼克松的白宫“敌人”名单，并成为乔治·麦戈文总统竞选团队的新闻副秘书；他后来就读于哈佛商学院以及哈佛大学肯尼迪政府学院，并最终回到了加州大学伯克利分校。他在那里接触到了伯克利的 Unix 文化，对他而言，这就像一场计算机技术的解放运动，符合他自由主义的政治观点。盖奇于 1982 年加入了脱胎于伯克利 Unix 文化的太阳微系统公司，此时该公司刚刚成立两个月，盖奇是第 21 位雇员。

近些年，盖奇一直担任太阳微系统公司的科学办公室主任。他的工作是周游世界，寻找技术发展的新动态和新思想。对于外界人士来说，他也是传播太阳微系统公司技术的使者。他是营销界的精英。

高斯林那天“同盖奇一起坐在车里”，心想这次推介又会是一场激动人心、近乎失控的冒险。不出高斯林所料，蒙特利会议中心的互联网连接一片混乱，而且并未经过测试。他花费了一晚上的时间，直到第二天凌晨 5 点才将网络连接恢复到正常有序的状态。大部分时间他都在通过电话与 MCI 的工程师进行沟通，指导工程师如何在会议中心的路由器上设置软件。第二天，盖奇发表了以科技对教育日益显著的作用为主题的演讲。在演讲的结尾，高斯林也出现在了台上。盖奇进行解说，高斯林则切进一张网页，上面是一个有着三维立体阴影的巨大的红色分子模型。这本身并不算新奇，但是，高斯林随后开始操纵那些组成分子模型的球状体，这着实令人耳目一新——高斯林点击并移动了几下鼠标，这些小球便开始旋转、弹跳。

理查德·索尔·沃尔曼是年度 TED 会议的组织者。他回忆说，盖奇演讲时听众的反响并不是很大。“当然，”他补充道，“后来每个出席了那场会议的人都会说，‘天啊，我是在那场会议上第一次看到那种技术的’。”TED[3]是好莱

坞与硅谷精英云集的一个盛会，参会人员包括电影制片人、演员、作家、出品人、音乐家和为数不多的技术人员。除了盖奇，像奥利弗・斯通或者昆西・琼斯这类艺术界大腕也会被邀请来 TED 演讲。但是对于那些 1995 年年初便对网络有相当了解的人来说（这类人在 TED 参会人员中并不是很多），盖奇展示的软件令他们感到震惊。高斯林回忆道，当他向其中几个人介绍这项技术后，“他们说‘天啊’，这改变了他们的世界观，至少他们对网络作为一种媒介表示了认同”。

当时，网络主要是用作在信息空间中储存大量文本和图片的图书馆。但是高斯林所展示的软件可以通过网络将程序传输给任一用户的计算机。它具有将网络从静止的页面媒介转化成互动式程序的潜力。他解释道：“就像你拿到一本书，翻开它，其中的页面都会与你讲话，你也可以随意移动书上的文字，让它们按照你的意愿排列。”这是一个生动的比喻，主要说明了软件对于将互联网最清晰可见的部分——网页——程序化的意义。更重要的是，从现代经济意义上说，将网络这种低成本的全球沟通媒介程序化，可以提高公司和客户的沟通效率、速度和多样性。这就是广义上的电子商务，而这还仅仅是一个开始。高斯林设计的这款名叫 Java 的软件是网页编程的主要工具，它为互联网创造了更多可能性，拓宽了其应用范围。这跟存储程序（stored-program）的概念有异曲同工之妙，因为程序存储技术扩大了计算机的使用范围，使计算机成为一种通用的机器，并可根据不同的用途进行编程。

现在，Java 已成为众多互联网编程语言的一种，是许多大学计算机课程的首选授课语言，也是深受年轻一代程序员欢迎的编程语言。从某种角度来说，Java 是互联网时代的 FORTRAN——由一家公司研发并拥有的编程语言，被程序员当做掌握未来的技术。太阳微系统公司有一个团队负责研发 Java，创始人是詹姆斯・高斯林。

一些人质疑高斯林的成就。因为 Java 反映的技术思想，早在 10 年前甚至 20 年前就在编程界出现过，所以，他们说 Java 并不是真正的新生事物。虽然

构成 Java 的技术要素在很多年前就被提出过,但是高斯林将它们真正组织在了一起，并形成了互联网通用的编程语言。所有的创新都是递增的，以原有知识为基础。但需要用创造性的观点对原有知识进行全新的组织。Java 体现了这种焕然一新的组织洞察力。拉吉・瑞迪指出："只有詹姆斯有解决该问题的独特能力，他对此有着深刻的见解，他的作品都很简洁、精炼。" [4]拉吉・瑞迪是卡耐基–梅隆大学计算机科学系的教授，高斯林在这所大学获得了博士学位。

Java 充满了计算机科学的思想，但它同时也是一种为了实用而设计的计算机语言。Java 设计中的机智和实用性完全体现了设计者身上的特质。高斯林是太阳微系统公司的科研中坚，也是公司副总裁，这是他特有的非正式头衔。他身材魁梧，高一米九，蓄着胡须，有些秃顶，栗色头发及肩；即便身着 T 恤衫和牛仔裤，看上去还是像一个上了年纪的维京战士——这是他在加利福尼亚州山景城太阳微系统公司研究实验室的标准装扮。

高斯林在加拿大的阿尔伯塔省卡尔加里长大，是家里三个孩子中最年长的。他的母亲乔伊斯是高中家政学教师。父亲戴维做过很多工作，从事过油田设备和房地产销售，曾一度跟建筑工人打交道，但是大部分时间处理的都是位于北部偏远地区的石油和矿产开采公司的物流工作。在高斯林的记忆中，父亲经常从像巴芬岛或者图克托亚图克那样遥远的地方打来电话，并且"电话那端的声音常常很飘渺"。对于高斯林来说，父亲的事业就是一个小创业者苦苦挣扎的故事。"父亲总是在不断地寻找生意，但是他从来不是家中挣钱最多的人，"高斯林说，"养家糊口主要还是靠我的母亲。"

父亲的四处奔波给高斯林留下了深刻的印象，给了他一种所谓的"创业过敏"。他解释道："我喜欢去创造产品并看着它们投入使用，但是说到创业，我就浑身难受，因此我并不像硅谷大多数人那样热衷于开创自己的事业。"

他最初的兴趣爱好是手工制作，修补以及复原物体的表面。他同邻近农场

的亲戚在一起度过了很长时间，尤其是在夏天。他回忆说他的祖父亚瑟“有个很大的废料场，里面有很多旧的农场设备”。大概从六岁开始，高斯林就发现自己被那些破旧的打捆机、脱粒机和拖拉机迷住了。在他 10 岁或 11 岁的时候，就已经对修理机器颇有心得。12 岁时，他制作出了自己的第一台计算机设备，Alberta 电话公司在无意中帮助了他。就像前青春期孩子的垃圾搜寻大冒险一样，他翻捡这家电话公司后面的垃圾桶，发现了一些手机继电器和开关。于是，他把这些从垃圾堆里翻出来的元件稍作改装，便用闪光灯、开关和胶合板组装起来，制作出了一种能够玩一字棋游戏的新玩意儿。后来很多程序员都是通过这款游戏接触到了逻辑树中的 if-then 和 and-or 基本运算问题。凭借这台机器，高斯林赢得了当地的科技展奖项。

高斯林的家距离卡尔加里大学只有一英里，正是这所大学让高斯林沉浸在计算机编程的海洋里。当父亲将他带进这所大学并走过一栋建筑的时候，高斯林注意到一些教室的门是上锁的。要想进去的话，人们需要输入一些安全代码。高斯林对此很是好奇，因此随后又来到这些门前，看着那些人输入密码并记下了这些代码序列。他回忆道：“对于这件事情我很在行；一旦你进去了，那些人便会觉得你本来就是那儿的人。”

大学的计算机中心就坐落在这栋楼上。十几岁的高斯林就像计算机房里的健身迷一样，是一个热切而充满好奇心的孩子，整天围着计算机转。这个孩子在计算机编程方面显示出了惊人的天赋。高斯林很擅长也很喜欢软件方面的工作，就像小时候喜欢祖父的农场设备那样，而且这种迷恋只增不减。他解释道：“从很多方面来讲，软件都是最复杂的发明。”

15 岁的时候，高斯林获得了一份物理系的兼职工作，通过编写代码来帮助分析加拿大 ISIS-II 卫星传回的数据。这颗卫星负责收集北极光对国际无线通信影响的相关信息。高斯林所参与的项目是加拿大太空计划中的展示研究项目，他回忆说，自己在其中的贡献就是为数字化设备微型计算机 PDP-8 编写“一大段一大段”逐行输出的代码。

高中时，高斯林经常逃课跑去大学的计算机实验室工作，为此常遭到学校领导训斥，却得到数学和科学老师的默默鼓励。他说："我是一个恼人的好学生。"在卡尔加里大学，高斯林依旧如此，他的专业是计算机科学，很明显他在其他课上的出勤率都不高。1977 年，高斯林毕业了，在计算机科学系主任安东·科莱恩的帮助下才获得了学士学位。科莱恩出面帮忙是不想让高斯林仅仅因为没达到一两门课程的要求就无法获得学位。

卡耐基–梅隆大学在录取高斯林攻读研究生学位的时候，常规程序再次为他作出了某种让步。高斯林共申请了 4 所计算机科学方面一流的美国研究生院校——斯坦福大学、麻省理工学院、加州大学伯克利分校以及卡耐基–梅隆大学，但是只有这所位于匹兹堡的学校录取了他。卡耐基–梅隆大学的这次录取有一些随意，因为高斯林是在一次平常的鸡尾酒会上被一位教授通知录取的，那时他刚刚完成论文《代数的操作性约束》的答辩。

卡耐基–梅隆大学每年从数千名申请者中挑选出 15 名学生；像其他学校一样，这种挑选在很大程度上基于知名计算机专家的推荐。高斯林毕业于卡尔加里大学，这所大学并不在知名计算机科学专业的版图内，因此高斯林明显处于劣势。在标准的录取程序中，高斯林并未被录取。但是该大学每年都会破例录取一名优秀学生，这个录取名额掌握在系主任约瑟夫·特劳布的手中。那年，这名幸运的学生就是高斯林，他在一次鸡尾酒会上得到了这个通知。"我并没有成功，起码在正常的录取程序中我没有胜出，"他说，"因此，从这个意义上说，我是中大奖了。"

1979 年，特劳布离开卡耐基–梅隆大学，成为哥伦比亚大学计算机科学系的主任，所以高斯林在 1983 年获得博士学位的时候，他并不在卡耐基–梅隆大学。18 年后，在俯瞰哈德逊河的曼哈顿公寓里，特劳布笑称，高斯林所谓的中大奖有点夸大其词。特劳布已经无法确切地回忆起当时的情况，他说可能是高斯林作为本科生的那种认真严谨的态度打动了他，"但可以肯定的是，"特劳布补充道，"能够被我们录取，确实需要好运气。"[5]

当时，高斯林这个成熟的年轻小伙子很快就赢得了软件奇才的名声，给卡耐基–梅隆大学的师生留下了深刻的印象。他的同学萨提什·顾普塔认为："他是全班最厉害的程序员，很可能是全校最优秀的程序员。"[6]据顾普塔回忆，入学的第一年，高斯林就开始做一些野心勃勃的项目。他写出的计算机程序有数万行编码，这些程序的庞大及复杂程度远远超过了一年级研究生的水平，但是对于高斯林来说，这只是一个周末就能搞定的事情。

作为一个明星程序员，高斯林肩负着将学校系统升级为 Unix 的艰巨任务。Unix 是贝尔实验室研发的功能强大、操作灵活的系统。贝尔实验室将 Unix 系统以相对低廉的价格授权给大学使用，并且对其应用和改进基本没有限制，因此 Unix 成为一些精英计算机科学教育院校的首选，其中包括加州大学伯克利分校和卡耐基–梅隆大学。这给年轻的研究者提供了充分的思考和创新空间。作为卡耐基–梅隆大学的 Unix 高手，高斯林经常与加州大学伯克利分校的 Unix 怪才比尔·乔伊互通邮件，而后者便是太阳微系统公司的创始人。

当数字设备公司强大的新一代微型计算机 VAX 于 20 世纪 70 年代后期生产出来的时候，卡耐基–梅隆大学决定在这台计算机上试运行 Unix，而不是该公司的新操作系统 VMS。当时学校的很多项目都在早期的初级微型计算机上运行，因此拉吉·瑞迪便询问高斯林，能否让之前的旧程序在新机器上运行。最简单直接的方法就是编写一个语言编译器，帮助逐行修剪旧机器上的程序，使之可以在新机器上运行。然而高斯林并没有这样做，他采取了一项更为大胆的措施，那就是创建一个机器语言转译器，这是一种更高级、更自动化的程序。它既可以解决当前的问题，确保为旧式微型计算机编写的程序可以在新式计算机 VAX 上运行，同时也为更多程序在不同的计算机上运行提供了通用的解决方案。

高斯林说，在卡耐基–梅隆大学的工作为他奠定了开发"Java 虚拟机"的

技术基础。顾名思义，虚拟机是一个像小型计算机一样的软件程序。程序员编写可以在虚拟机上运行的应用，虚拟机随后转译这些编码，使之在不同的机器上甚至不同的操作程序上运行。

20世纪70年代末，高斯林开始思考独立于计算的软件。“很明显，詹姆斯的脑海里一直都有这些想法，”瑞迪说，“正是这些想法帮助高斯林制造了小型压缩式虚拟机，可以通过编码将其传至整个网络。这就是Java问题——怎样编出一种流传于网络、可以在任意机器上运行的通用编码。詹姆斯围绕着这一想法思考了很多年。”

高斯林曾离开卡耐基–梅隆大学一年的时间，他说这样做的原因主要是要远离一份“变质”的关系。他搬到旧金山，为硅谷的公司做咨询工作，主要是电子邮件系统方面的内容。返回匹兹堡之前，他获得了博士学位，而后开始寻找一份全职工作。

当时，太阳微系统公司刚刚起步。高斯林的两位备受Unix文化熏陶的朋友——伯克利的比尔·乔伊和斯坦福的安迪·贝希多斯海是这家公司的创始人。他们试图劝说高斯林加入公司，但是都被拒绝了，一部分原因可能是高斯林的“创业过敏”。与此同时，IBM也向他伸出了橄榄枝，请他为工作站编写软件程序，以供工程师和科学家进行计算机辅助设计和研究之用。这也是太阳微系统公司想要开拓的领域。“IBM公司有一群很酷的员工，他们做出的原型机全面摧毁了安迪等人的想法，”高斯林回忆道，“我当时几乎可以肯定太阳微系统公司将在几个月内倒闭。”

在IBM的工作中，高斯林从一个普通员工的视角见证了先进的技术是怎样被政治和官僚主义所压制的。这个工作站成为了企业规模冲突的副产品，高斯林说：“工作站的规模宏大，成了一个不小的阻力，我简直不敢相信他们如此愚蠢。”在IBM工作一年之后，高斯林来到硅谷并加入了太阳微系统公司。1984年，抵达硅谷不久，高斯林开启了一项富有冒险性的项目，那就是NeWS（Network Extensible Windowing System，网络可扩展窗口系统）。他的想法在当

时相当具有开创性——网络上的任何一台计算机屏幕都可以显示其他计算机正在运行的程序。其他软件专家起初对这种想法十分怀疑，但是 NeWs 成功了。这与高斯林不遗余力地寻找方法来凝练编码和加快程序运行速度有关，这是实现移动代码的必要条件。尽管 NeWS 在技术上是成功的，但是它从来都没有成为一个商业化的产品。不过，它表明了网络中代码转化技术的潜力，高斯林通过 Java 将这一潜力变成现实。

Java 出现于 1990 年。年轻的程序员帕特里克・诺顿宣称他计划退出太阳微系统公司并加入 NeXT 计算机公司，后者是史蒂夫・乔布斯退出苹果公司之后建立的工作站。诺顿逢人便讲 NeXT 是最热门的科技公司。太阳微系统公司的主席斯科特・麦克尼利同意这一看法，他请这位年轻的程序员来为他写一个备忘，希望他指出太阳微系统公司错在哪里。诺顿言辞犀利的邮件在太阳微系统公司内部广为传阅。虽然这并不是下一步行动的蓝图，但是它却是太阳微系统公司内部大讨论的一个催化剂。它触动了公司的技术神经，让大家意识到了太阳微系统公司正在停滞不前，需要从大局思考，创造出符合新型计算机市场发展的优秀产品。在麦克尼利的带领下，公司成立了一个小组来完成这一目标，小组成员包括高斯林、诺顿（他被说服留在太阳微系统公司）等人。

这为太阳微系统公司提供了长远的发展契机，他们着眼于未来十年的发展。技术发展的大趋势一目了然。根据摩尔定律，微型芯片将稳步发展，这意味着不久之后，数字化电路板、计算能力以及一些智力元素将应用于各种设备。但是只有在计算机可以通过网络进行沟通的情况下，这种普适计算才能实现。太阳微系统公司认为，第一个不错的机会就是消费电子产品，数字技术已经在此方面崭露头角。他们做了几个月的功课，终于有了激动人心的发现：索尼、夏普、松下、三菱电子等大型企业都认为消费电子产品和计算机产品两大领域正在不断融合。这些公司都格外当心微软和英特尔，因为这两家公司控制着个人计算机业务的软件和硬件标准，是与消费电子产品关系最密切的计算机领域公司。这些大公司担心会成为微软和英特尔的技术人质，它们并不希望这件事

情发生。

对于太阳微系统公司的小组来说，下一步要做的是开发一些原型技术并且制订商业计划。几乎从一开始，这个工作小组就搬出了公司园区；他们先是入驻帕洛阿尔托一家风险投资公司中几间没有门牌的办公室，1991 年春天又搬到了门罗帕克一家美国银行支行楼上宽敞一些的办公室。他们的工作是保密的，太阳微系统公司的其他员工甚至需要签署保密协议才能来访。当时的团队负责人迈克尔·谢里登回忆道："现在看来这种举动有些愚蠢，但是可以肯定的是，当时公司的一些'抗体'得知这件事后一定会杀了我们，或者让投资者撤出资金。"[7]他们早期的产品研究包括长时间玩任天堂游戏。不久之后，他们制定了一个目标：为消费电子产品设计一款掌上管家。"有了它，人们就可以控制电视、摄像机、立体音响等与其无线网络相连的设备。它要有一个触屏，触屏上有很多设备和信息源的图标，比如电视指南。如果你想用 VCR 录制电视节目，只需将电视指南列表拖至电视机的图标上，然后松开。它还可以相互连接，例如，如果你的母亲需要帮助，她可以通过邮件将 VCR 的操纵装置发送给你。没有晦涩的指令，没有键盘，只需要开机动动手指就可以了。高斯林要做的就是研发底层软件工具，让其变为现实。"

当时，这个项目叫做 Green。[8]它最初的商业计划"绿色大门背后的秘密：关于消费电子产品机遇的深入思考"颇有先见之明，至少在在软件需求方面催生了 Java。（至于为什么要用这样一个标题，高斯林回忆说，他们的办公室都有一扇绿色的大门，他们都在这扇门后秘密地工作。这个标题与 20 世纪 70 年代一部著名的色情电影相对应，"体现了我们的幽默"。）1991 年 8 月 23 日，一份标有"太阳微系统公司机密财产"字样的文件描述了发展中的技术、重要性及其市场潜力，长达 44 页。这份报告将消费电子产品广泛地定义为"包括 VCR、电话、游戏、手机、洗碗机以及调温器在内的产品"，Green 将会提供技

术“使这些设备能够透明地交互操作”。报告还指出，Green 是一个“软件科技开发团队”。

这份报告说，太阳微系统公司会很快成为消费电子产品的标杆性企业，正如个人计算机产业中的微软公司一样。“这是一个‘打破’微软进入消费电子产品行业的绝佳机会，因为这个行业的标准和界面都是不固定，尚未定义的。”

在这个计划中，有一段关于 GreenTalk 的描述，这是一个“健壮的编程”语言。报告用两页小号字体论述了 GreenTalk 是处理分散移动代码的理想语言，列举了现在 Java 的某些特性，包括“动态适应”不同机器环境的虚拟机器概念。多年后高斯林说道：“Java 研发涉及的大量工作都来自那个商业计划。”

在开展 GreenTalk 计划的那些日子，高斯林可能一直在考虑消费电子产品市场，但是他所强调的问题与 Java 反映的互联网问题惊人地相似——怎样将改进软件，使其通过庞大的网络将程序传输给客户，从而让客户使用各种各样的计算设备来运行该程序。他在构思能够在复杂多样、充满不熟练用户的计算机世界，依靠自身发展壮大的网络计算，而不是像 1991 年的私有企业网络那样，只供专家使用。“一旦你走进网络化的计算机世界并与真人展开交流，一切都会自然而然地发生，”高斯林说，“设计 Java 的所有想法都来自解决这个问题的过程。”

高斯林的这些想法是随着时间慢慢形成的。在 1990 年这个项目开始的时候，他并没想过自己会创造一种全新的编程语言。他回忆道：“它开始只是改进 C++语言的修补代码。我当时编写这些代码只是想使掌上样机项目成功。但是我越深入，就越清楚我需要更加深入。”

他回忆说，1991 年的某一天他突然清楚地意识到新机器语言的必要性。当时正值杜比兄弟合唱团在山景城开演唱会[9]，高斯林的妹妹芭芭拉得到了两张前排的座位。高斯林记得当时扬声器就在他后面一点，所以实际上他可以听到歌手的原声。坐在舞台旁边，高斯林盯着舞台的灯光系统看，这个系统控制得很稳定，随着音乐旋律的变换而变化。娴熟的灯光系统控制看上去与太阳微系

统公司团队的理论研讨很相似。高斯林说，“这使我开始思考，”要解决网络环境安全问题，新软件的设计是必要的，“不是增加一些东西就可以彻底解决问题，而是需要深入最底层。”

“绿色大门背后”计划正是底层的一根标桩，是软件的一个发展方向，而非完成品。1992年，高斯林用了一整年的时间来研发新语言，这就是后来被他称为 Oak 的语言。名称源于他办公室窗外的那棵橡树。这种掌上样机 Star7 于 1992年9月完成，并送至麦克尼利面前。这个触屏设备让人印象深刻，它有着时髦的图标和声效，还有一个虚拟向导 Duke，它是个有着红色大鼻子的卡通人物。麦克尼利欣喜若狂，太阳微系统公司设立了一个全资子公司 FirstPerson 专门进行营销。消费电子产品公司也被深深地吸引了，但是它们认为投入设计的科技成本太大，产品不太可能大众化。因此他们并不看好这个产品。但是这种否定不久之后便被人遗忘了，因为，又一个机会到来了。

截至 1993 年年初，传媒、科技和娱乐产业一直都被“信息高速公路”这一概念牵着鼻子走。电视将会重新数字化——500 个频道、点播电影、电视购物服务、报纸、杂志和书籍都将通过屏幕展现出来。这年春天，时代华纳公司声称正在奥兰多寻找互动式电视试验的合作伙伴，这掀起了一股热潮。科技公司相互竞争，希望可以为时代华纳公司提供机顶盒，而其他有线电视公司也匆忙宣布了自己的计划。

包括高斯林在内的 FirstPerson 集团在接下来的几个月里试图说服时代华纳公司，太阳微系统公司将是最好的机顶盒供应商，它使用 Oak 作为理想的软件中介来处理所有的信息，提供娱乐和电子商务支持。最后这份合约花落硅谷图形公司，高调的奥兰多互动电视试验和其他相关的试验最后以失败告终。

在竞标奥兰多试验的那几个月里，太阳微系统公司的团队与有线电视公司高管在概念性问题上的认识鸿沟深深地困扰着高斯林。长期以来，高斯林和同事们都认为信息高速公路就是“一个为其他人服务的网络”，而不是像 20 世纪 90 年代初兴起的计算机网络那样，仅适用于计算机高手。在他们看来，网络将

会开放形形色色的服务，而且主要由客户来控制。“我们谈论的是一种完全的平均主义，大家是对等的，”高斯林回忆说，“他们的技术人员喜欢我们的提案，但是那些有线电视公司的高管并不喜欢，因为他们的生意主要是要吸引眼球。这些人有很强的控制欲，而我们宣扬的是反对控制。”

FirstPerson 在 1994 年春天宣告破产。受这件事的影响，高斯林的事业也面临着危机。事实上，高斯林事后被告知，当时有一个计划要取消 Oak 项目并且要将这些程序员派往他处。但是这个项目没有实施，这在很大程度上多亏了比尔・乔伊。作为太阳微系统公司的创始人，乔伊是公司的技术领袖和董事会成员。他不仅是一位优秀的计算机科学家，还是一个果断的天主教知识分子。乔伊的聊天话题非常广泛，且具有跳跃性，从歌剧、文学、建筑到曼哈顿各个酒店的优势，最后才会回到编程这个正题上来。[10]1991 年，乔伊从硅谷搬到了阿斯彭，据他说这是为了“丢下迫切的事，去做重要的事”。他在阿斯彭创办了太阳研发实验室，解释说这个小镇有着极好的便利设施，包括优质的滑雪场和一流的书店。

如一位同事所称，乔伊在 Oak 项目上从一开始就是一个“调皮却有贡献的人”。他通常会拿几周时间专注于这个项目，然后在随后的数月中消失不见。但是只要在这个项目快要被取消的时候，乔伊就会出面干涉。高斯林说：“比尔在这件事上做了很多严肃的抗争，为这个项目的复活奔走呐喊。”

1994 年，互联网风潮开始盛行。1993 年 6 月，两名伊利诺伊大学国家超级计算机应用中心的学生马克・安德森和埃里克・比那发布了第一个版本的 Mosaic 浏览器。为了使这个浏览器商业化，安德森于 1994 年 4 月创办了网景通信公司（Netscape Communications Corporation，最初称为 Mosaic 公司），比那也随即加入。虽然不能立竿见影，但是使用易于操作的软件浏览网页，有助于使蒂姆・伯纳斯–李发明的网页从供研究人员使用的工具变成大众传播媒介。网络提供了展示、识别和链接网上信息的技术，而浏览器增加了易于使用的浏览程序，这使得所有个人计算机用户均可访问网络。高斯林所发明软件的

适用性很强，可用作强大的互联网编程语言。在塔霍湖的一个休息寓所，乔伊指出了这一点："我告诉他们'游戏已经开始了'，这个游戏就是互联网。"在这一点上，高斯林等人表示同意。"我们认为互联网并不会以消费者为导向，"他说，"但是因为网页和浏览器，互联网已经开始这样发展了，所以我们只能顺水推舟。"

乔伊一直认为，根据太阳微系统公司的商业准则，高斯林设计的语言即使不直接与能够产生利润的硬件产品挂钩，本身也是计算机界的一个重要发展。从1970年末起，在修改加州大学伯克利分校版本的Unix时，乔伊就已经开始思索，如何在计算机网络层出不穷的环境中用一种编程语言确保网络使用的安全性和可靠性。这需要将几种先进的计算机科学融合在一起，转化成一种新的语言。"我并不知道如何做到，"乔伊说，"但是我一看就知道了。"Java就是要找的编程语言。

在Java的设计中，乔伊认识到高斯林作为程序员的超凡天赋。"詹姆斯很伟大，因为他在这个空白的领域写下了第一笔，虽然只完成了一部分但异常精彩。他独自一人完成了这项工作，这是一种罕见的技能。要将这项技术飞速应用到产品中去，其他人就必须要效仿他。但是像詹姆斯这样的先行者寥寥无几，毫无疑问，他有着独特的思考方式。"

太阳微系统公司前技术主管埃里克·施密特指出，Java的天才之处在于"詹姆斯的借鉴和创新恰到好处。"[11]

充分领会高斯林在软件设计方面的借鉴和创新，有助于理解Java产生的技术背景。这一背景取决于Java的发展方向和发展阻碍。高斯林的目标是建立一个网络化的世界，其中各种软件在日益复杂的互动模式中相互融合、碰撞和交流。这与非网络化的设置截然不同，在非网络化的设置里，各种程序就像位于穷乡僻壤的独立部落。但是软件一旦接触更广泛的互联网世界，就失去了孤立

部落的安全性。在相对独立的环境中无害的程序漏洞，却可能给网络世界带来无法预见的不良影响。恶意程序即病毒，生动地展示了故障在网络中传播的可能性。

Java 发展的阻碍也包括用于工业编程的主流的 C 语言及其继承者 C++语言。高斯林及其团队使用 C++语言作为向导，来判断在某些方面应该注意的问题。C++语言给予程序员很大的自由，来操纵数据并进行比特级编程。它是一种强大的工具，但是使用起来也很复杂，对于非专业程序员而言，这种编码语言也很危险。亚瑟·范·霍夫说："C++语言就像 M-16 一样。"[12]他将 C++比喻成军用自动来福枪。荷兰籍编程高手范·霍夫于 1992 年加入高斯林的团队，他注意到这个团队从一开始就专注于构建一个"没有锐利边缘的编程语言，这样人们就不会因为编程语言的问题而伤到自己。要知道，90%的程序员都是技术水平一般的普通人，所以在网络化的环境下，复杂的计算机语言就像是一个杀手。"

高斯林提供了一种全新的观察视角。1972 年，贝尔实验室创造了 C 语言，旨在让程序员对计算机有更好的把控能力，因为当时的计算机还处于小内存时代。"C 语言在那个时代是一个很伟大的设计，"高斯林说，"但是代码使用者所关心的问题是不断改变的。当时的计算机不是通过网络进行连接的，而是关在屋里彼此隔绝。没有人关心不同的计算机上使用何种编程语言，只是希望和祈祷自己的编码能够在面前那个又大又慢的计算机上顺利运行。"

相反，Java 是一种互联网时代的语言。设计它就是为了在复杂的网络时代简化程序员的工作。Java 首先通过设计一些工具来使程序员的工作更加简化和高效。其中一个工具就是自动化存储管理，或者编程行话中所谓的"垃圾收集"。垃圾收集是一个软件片段，它搜索计算机内存中不起作用的数据和编程片段，将它们清理出去，为紧迫的工作腾出存储空间。C 语言和 C++语言并不具备垃圾文件收集工具这一标准特点，这就意味着程序员必须手动控制计算机的内存管理，制作清单并记录下无用的数据。高斯林认定，这是一件耗时耗力的工作，

程序员因此没办法从事更有意义的工作。这种人工手动的模式也经常导致软件运行失灵。此外，由于计算能力大幅提高，运行垃圾文件收集程序并不会对计算机运行速度产生很大的影响。因此，20 世纪 90 年代，在高斯林进行 Java 设计的时候，那些早先存在缺陷的设计给很多用户带来了不便。高斯林指出：“20 年来的摩尔定律带来了巨大的变化。”

Java 也试图通过限制程序员犯错的自由来简化互联网编程。Java 中存在很多限制，其中很多是深层次的技术问题，但是一些例子表明这正是 Java 所反映的文化。例如，Java 对于“异常”或者说背离常规的编程要求相当严格。在编译软件时，程序员可能会对“异常报告”或者程序运行之前的警告很警觉。

在其他语言当中，这些警告通常只是建议，可以忽略。事实上，这些异常并不一定会产生问题。但是，在网络时代，一个运行古怪的程序很可能引起麻烦。Java 要求程序员对这些异常警告作出反应。该语言迫使程序员去思考程序中可能出现的故障。这种努力会让软件在复杂的环境中，尤其是在多变的互联网环境中更加可靠。

Java 的一个显著特点是，它是被计算机科学家誉为“具有鲜明风格”的语言。简单来说，这意味着数据的种类和运行都是严格分类的。在 Java 中，软件模块都是被严格定义的，它的界面、连接也是如此，这要求它们只与软件的特定区域进行互动。

乔伊说，互联网世界的最终目标是努力将软件变得像机器的组成部件一样稳定。Java 可以被视为装有螺丝钉、螺丝帽、螺栓、撬杠以及各种用具的工具箱。Java 的鲜明风格就是一个系统，它确保程序员在使用这些工具的时候不会做出什么危险的举动。确切地说，Java 是一种“系统”语言，就像 C 语言、C++语言和 Lisp 那样，它使专业的程序员通过比特级控制深入了解机器，并且最终改变计算机的基本操作。不过，Java 也强制执行结构化准则，没有其他路径可选。它在规避风险的同时，也夺走了一些编程的自由。

并不是所有人都能接受 Java 对程序员的这种限制。那些厌恶这种限定的程

序员将 Java 视为“警察国度”，而高斯林就是软件法西斯主义者。高斯林回应说，Java 的纪律性对于整个互联网环境而言是一种宝贵的资产，它将程序员从一些琐碎的具体工作中解放出来，并且让软件变得更加稳定可靠。有一点需要承认的是，Java 的确是折中的结果，但是利远大于弊。

“对 Java 而言，规矩就是规矩，”高斯林说，“一旦你适应了这一点，它将会真正变成一种自由的语言。”他将对 Java 的批评比作早期的飞行员公开抗议。当飞机制造商将驾驶舱密封起来的时候，飞行员们提出了抗议。因为在使用螺旋桨推动飞机的旧时代，飞行员是通过将脑袋伸出机舱来导航、嗅空气，从而感知风向和天气的。“但是，如果你坐在一架马赫数为 3（3378 千米/小时）的飞机上，打开机舱将脑袋伸出去，你的脑袋将被吹飞，”他以这个例子来佐证他的观点，“要更进一步解放自己，你就要放弃那些曾经看似自由的东西。”

到 1994 年，高斯林已经确立了主要的概念，但是要想将 Java 变成互联网编程的工作语言，他还有很多工作要做。尽管最终证明 Java 生逢其时，赶上了互联网浪潮，但是最初这并非显而易见。成功的技术总是在事后才被看做必然。网络成为商务和沟通的主流媒介的变革才刚刚开始，Java 可以满足互联网编程语言发展要求的趋势尚不明显。

1994 年秋，盖・斯蒂尔从思维机器公司（Thinking Machines）跳槽到太阳微系统公司。思维机器公司是一家富有创新精神，但在商业上止步不前的超级计算机制造商。斯蒂尔毕业于麻省理工学院，是著名的计算机科学家、编程语言和人工智能专家。加入太阳微系统公司之后，斯蒂尔针对公司的研究项目做了调研。公司正在开发很多“取代 C 语言”的编程语言，包括之后命名为 Java 的 Oak。斯蒂尔认为 Oak 是一种严格的书面语言，它抽离出很多 C++语言的元素，并且加入了垃圾文件收集程序和一些其他工具。“但是我当时并没有太看好 Oak，”他回忆道，“直到后来我才看出它的实用性。我有个盲点。”[13]

在接下来的一年里，公司制定了一系列关于 Java 的决策，包括 Java 要有什么特色，需要抛弃什么，怎样精确地呈现每一个特色，以及怎样将这些特色整合在一起。但是，这些编码需要进一步的修正调试才能够改善性能。那些至关重要的决定是在硅谷和阿斯彭召开的一系列会议上作出的。会议只有六七个人参与，讨论的问题也许是技术层面的，但是围绕问题的讨论确实激情四射，有时候甚至针锋相对；詹姆斯·高斯林和比尔·乔伊之间的辩论尤为激烈。作为技术主管，埃里克·施密特也参加了几场在阿斯彭召开的会议。“他们态度很强硬，让人难以置信，有时候你会觉得很不舒服，”他回忆道，“但是这也可能只是我个人的一些感受，这就是那些意志坚定、聪明卓越的人的工作方式。”

高斯林与乔伊辩论的核心是编程语言的简洁性和功能性。乔伊的观点是研究团队应该在 Java 中融入一系列前沿性的语言特色，尽量使 Java 变得强大。乔伊解释道，几十年来，出现了一代又一代的计算机语言，C 语言出自 20 世纪 70 年代早期的贝尔实验室，但是其“祖先”可以追溯到 20 世纪 60 年代研制的 BCPL。一旦一门主流语言奠定了其地位，那么想要动摇它就很困难了。

乔伊从 Java 中看到了一个让计算机语言数字化、系统化和符号化的机会。为了解释这种观点，乔伊举例说明了人们将如何使用越来越强大的掌上计算机：对于股票价格和科学数据而言，类似 FORTRAN 的数字化语言是完全适合的；对于显示和网络连接，计算机设备应该使用像 C 这样的系统性语言；对于那些设计日程或者提出购买建议的直觉性任务来说，计算机设备应该使用像 Lisp 这样的符号语言。乔伊想把这些语言的强项都纳入 Java。“我将 Java 视为未来 30 年到 40 年的唯一一次机遇，”他回忆说，“因此，我试图将一切都纳入其中。”

高斯林是简洁性的倡导者。他的思想可以概括为“只要存在疑惑，就剔除它”。他指出，降低复杂度将会使 Java 更富有连贯性，更加小型化，更加友好，这样就更易于被程序员理解和掌握，会有更多程序员选择使用该语言。编程语

言越简单，人们掌握起来就越容易，这在 Web 和 Netscape 浏览器的快速普及上表现得尤为突出。按照高斯林的说法，乔伊扮演了一个很有帮助的参谋角色，在一些困难点上是个强有力的推动者，有几次甚至有些独断专行。“但詹姆斯是最终的权威，”会议常客盖・斯蒂尔说，“Java 就像是他的孩子。我们都认为对于编程语言来说，技术一致性是很重要的——语言的设计应该是一个人的观点。”Java 不应该是一个委员会式的编程语言。“回过头来看，”斯蒂尔补充道，“大部分正确的决定都是詹姆斯作出的。他在决定孰去孰留的问题上作出了很有鉴赏力的选择。他所作选择的重要性最终得到了证明。”

这种选择体现了一位计算机科学家务实的特质。早在卡尔加里的农场修理农具时，他就体现出了工程师的特质。当被问及一门成功的编程语言（不仅是 Java，还有其他几种语言）是怎样炼成的时，他承认其中有一些运气的成分。但是高斯林指出，每一代主流编程语言都是那个时期最有用的工具。“在语言设计界中，有很多学术讨论和争执，就像辩论有多少天使在针尖上共舞一样无聊，”他指出，“而我的拙见是，这些辩论要去解决现实问题，并且必须以不改变其他事情为前提。”

高斯林精明务实的举措之一就是让上千万名程序员很快熟悉了 Java。源代码——语言的最高级别，人类可读——被转换成了 C++语言。所以对于世界各地的编程大军而言，Java 看起来是一个可迅速掌握的语言，只需要在历尽艰辛学会的 C++语言基础之上稍微改进即可。这种表象在某种程度上是一种误导，但它的确加快了 Java 被接受的速度。像 C++语言那样，Java 由很多被称为对象的小软件模块或者构件组成。但是 Java 也吸收了其他语言的优点，诸如垃圾文件收集程序、字节解释器（虚拟机支持技术）以及严格的数据分类定义和行为准则等。

“对于开发者来说，Java 看上去像是 C++，”高斯林解释道，“但是，其实 Java 还大量吸收了 Lisp、Smalltalk 以及 Pascal 的特点。我想要的就是将这些环境平滑地组合在一起。”

正当 Java 成型之时，又一个商业计划出炉了。虽然在 1994 年春起死回生，但 Java 项目还是缺乏方向。高斯林、乔伊等人都在思考：互联网是当前的市场机遇，但是如何去发掘呢？在当年 9 月份召开的一次战略会议上，首席技术官施密特和乔伊会见了高斯林以及 Java 团队的其他成员。会议的主题是 Java 项目的目的与目标。当施密特提及这个问题的时候，高斯林的回答是他想要“所有的人都使用这种语言”。

战略会议结束之后，施密特立即起草了商业计划书，他接受了高斯林的目标，并且提出了具体的数字——在接下来的 5 年内，Java 软件用户要达到 1 亿。1 亿这个数字被当做“所有人”的基准，据施密特回忆，因为这是当时运行微软 Windows 系统的台式计算机数量。事实上，Java 用户达到 1 亿只用了两年的时间。太阳微系统公司能够取得如此成就，可能是因为它使用了对于新编程语言而言最激进有效的市场策略。这场营销活动不仅仅是要争取程序员，还要征服公司高管、新闻媒体，甚至公众。旨在为 Java 制造一种“必然性光环”，这样整个计算机产业——甚至微软公司——都将承认并使用 Java。

最重要的一步是赢得网景公司的认可，将 Java 嵌入其浏览器中。起初，网景公司的创始人明白浏览器只是第一步，它使检索和查看含有图片和文本的网页更加容易。但是如果要使网页成为商务和娱乐的日常媒介，就需要用到计算的真正实力了。网景公司的创始人之一马克·安德森说：“我们需要找到一种使网页可编程的办法。”[14]

事实上，针对这个问题，安德森和其他几位年轻的程序员已经思考了几个月：要把大型中央计算机或服务器上的移动代码通过网络传送到数百万台计算机上，什么才是最佳编程工具呢？为此，他们试图对 Lisp 及其变种 Scheme 加以改进，但是这些语言都是专用语言，很难阅读和理解。安德森及其团队随后看到了 Java，便认定了这种语言。Java 看起来具有必要的技术元素，并且让人

觉得似曾相识。“采用 C++语言的语法非常明智，”安德森说，“这意味着有很大机会让编程界接受它。”

施密特和乔伊与网景公司进行了协商，并于 1995 年 5 月 23 日签订了意向书。此后，工作便拉开了帷幕。高斯林设计的 Oak 改名为更加市场化的 Java。高斯林回忆了筛选名字的过程。他和项目组在一个放置有白板和会议设备的房间举行会议，会议的主要引导性问题是：这项技术带给你的感觉如何？3 个小时后，他们讨论出了十几个候选名字。位列第一的是 Silk，但是高斯林很讨厌这个名字。排名第三的是 Lyric，这是高斯林本人最喜欢的名字，他说，因为编程语言就像歌词一样，是“表达自己的一种方式”。排名第四的是 Java，因为有人说这个词语让他感到很兴奋，就像喝了浓咖啡一样。高斯林耸了耸肩说：“这就是联想的范畴。”他们把这些备选的名字送到太阳微系统公司的法律部门做商标检测。

不久之后，在由施密特主持的公司员工会议上，吉姆·波利思提出了两个候选名字：Silk 和 Java。施密特告诉新编程语言的产品经理波利思，由她来决定。她作出了自己的选择，Java——像高斯林窗外的那棵橡树发出的遥远声音。

对于太阳微系统公司而言，Java 的互联网商业计划是一个起点。乔伊和施密特负责实施这一策略。麦克尼利让他们放手去做，只需定期向他汇报工作。麦克尼利知道他们跟网景签订了意向书，并同意了。但是乔伊和施密特并未告知太阳微系统公司的管理层和董事会，他们的计划不包括从 Java 中获利，起码不是直接获利。授权条款全是无偿奉送的，只为了提高该软件的接受度。由于用 Java 编程的应用在所有的计算机和操作系统上都能运行，因此，这似乎对太阳微系统公司和微软公司都带来了不利。太阳微系统公司与微软公司一样，通过技术——运行 Solaris 操作系统的 Sun 计算机——“锁定”客户，来确保公司有丰厚的利润。

Java 战略就是一个赌注，打赌太阳微系统公司将因建立 Java 这一国际标准而获利。这次赌博后来获得了成功，但是这更多的是一种信念而不是常规的商

业计划。施密特回忆，在他与太阳微系统公司的高管针对 Java 计划进行非正式探讨时，他得到的回应既有怀疑也有彻底反对。乔伊也得到过不太正面的回应。他说："向一个组织引入一种新秩序是最难的事情，尤其是当其专注于另外一个模式的时候，这是马基雅维利的经典思想。"但是当 1995 年 6 月乔伊和施密特将 Java 计划详细展示给斯科特·麦克尼利时，他当即表示很喜欢。"斯科特理解了所有的含义，并且制订了策略，"施密特回忆道，"比尔·乔伊和我想出了这个计划，但是斯科特将其传达给了整个公司。"

太阳微系统公司自此之后开始了漫长的征程，为了将 Java 变成一种编程的行业标准，他们要构建一个新的商业化生态环境，促使越来越多的程序员用 Java 编程，写出越来越多的 Java 应用程序。到 2001 年，Java 已经得到 200 多家公司的许可，全世界有数百万软件开发人员在使用 Java。很多人相信在再过二三十年，Java 将成为占据统治地位的系统编程语言；但并不是每个人都同意这种说法，尤其是微软公司。

2001 年 1 月，微软公司与太阳微系统公司之间的长期官司尘埃落定，微软同意支付太阳微系统公司 2 千万美元，并终止其使用 Java 的许可。在这场诉讼里，太阳微系统公司指控微软违反了授权条款，因为微软公司在 Java 上增加了一些特殊的扩展，以服务于其 Windows 操作系统。太阳微系统公司宣称，微软公司试图"污染"Java，通过破坏 Java 可用于不同操作系统的基本原则，打破其标准。微软公司回应说这样做并没有错，并声称太阳微系统公司的这场诉讼是削弱微软的竞争手段。无论如何，官司了结之后，微软公司就宣布在微软的技术中使用一种"Java 用户迁移路径"，并且推出新的编程语言，C#或 C-sharp，用于替代 Java。微软公司的 C#语言定位与 Java 类似，有着与 Java 类似的功能，但是对于程序员来说没有那么多的限制。这是一本微软的策略教科书——引入一种技术，用自己的方式重新定义，然后说服数百万名程序员加入自己的阵营。"我们认为，另一种语言诞生的时机成熟了，"微软资深设计师安德斯·海尔斯伯格说，"在 C 语言家族中，我们已经有了 C、C++和 Java，那么有什么理由

认定 Java 是最后一个呢？”[15]

海尔斯伯格可能是对的，微软公司的 C#语言是一个强悍的竞争者。但是时至今日 Java 还是遥遥领先。越来越多的大学将 Java 作为授课语言。它成了互联网经济时代码农们的首选编程语言，正是这些程序员构建了交易系统和产业网站，推动了在线商业的发展。里奇・布坎南是纽约一家公司的工程总监，他们负责为大型公司建设并维护网站。在问及他的员工队伍时，布坎南总是这样回答：“他们是坚定不移的 Java 程序员，现在这些软件工程师都愿意使用 Java。这让他们的产品更有市场。Java 是掌控未来的技术。”[16]

Java 也给了人们一些启示。优秀技术、时机、运气和市场营销对于它的成功都至关重要。这个过程起起伏伏，时好时坏。但是太阳微系统公司之所以能够研发出 Java，是因为它集合最好的工程师构建了一个小型团队，让他们放手去探寻新兴的、重要的事物，并最终听从 Java 创始人詹姆斯・高斯林的判断。他不仅有话要说，而且他知道要怎么说，所以人们认为他对技术的主张是一种坚持，而不是固执，更不是妄自尊大。“你必须学会怎样向更广泛的受众解说，”他说，“你不能用太深奥的方式来解释，因为没有人能听懂。封闭式思维是工程师的弱点。”

然而，高斯林也是个毫不掩饰的“极客”。他沉迷于机械和对事物运行原理的探索。在开车去吃午饭的路上，高斯林很喜欢摆弄他灰色沃尔沃敞篷车里的新导航系统。它不仅在数字屏幕上跟踪汽车的运行，提供向导，还与数据库相连，可以将周围餐馆的分类，有法国餐馆、泰国餐馆等。高斯林笑嘻嘻地说：“现在，我有些饿了。”他并不看机器上显示的就餐建议，而是直接驶向了他最喜欢的那家印度餐馆。

高斯林说，天才程序员都有极客的基因。“这种基因有奇怪和痴迷的一面，”他解释道，“那些最优秀的人都有一种气质，他们会像中了魔一样被一些

事深深地吸引，全身心投入；他们也不知道为什么会这样。”

随着 Java 的流行，高斯林又回到实验室，致力于研究一些新型的软件开发工具，他称为“做手艺活儿用的锯和锤子”。高斯林所要推广的工具将会帮助精英程序员战胜软件的“复杂性危机”，因为互联网上会有越来越多的程序共同运行，相互配合。他现在的工作与 Java 扮演的角色相似，致力于研究制造产品所需要的软件工具。“我的整个事业就是这个样子的，”他说，“开始想制作一些东西，而后我发现要是有一些工具就好了，于是我开始制造工具。很快，你会发现自己在工具的泥潭里越陷越深，最后全然忘了最初的动机。”

第 11 章

一定有更好的方式：Apache 和开源运动

1994 年下半年，一小群分散在英国、美国内布拉斯加、旧金山以及世界其他地方的互联网软件工程师感觉自己被抛弃了，大家在网上同病相怜。在互联网进入千家万户之前，在华尔街发现网络价值之前，在浏览器战争之前，他们中的大多数都在经营网站。当时，互联网还是个要求严格的狭窄领域，以 20 世纪 60 年代末互联网发展初期的研究为指导。按照学术研究的传统，互联网软件都是免费共享的。

但是事情开始发生变化。美国国家超级计算机应用中心位于伊利诺伊大学，该中心的编程小组迁往硅谷寻找出路，加入了一家新成立的公司，也就是后来的网景公司。之前，那些同病相怜者都使用由伊利诺伊超级计算机应用中

心开发的数据服务软件，将网页通过互联网传送到台式计算机上。随着伊利诺伊开发者的离去，这一大批网络专家就只能以非正式的形式在内部分享代码改进和漏洞修复。这群人中的一员布莱恩・贝伦多夫说："我们像交换棒球卡一样交换软件补丁。"[1]

但是，由于缺乏领导和规范的方式，大量的快速修正很快导致程序布满混乱的补丁。因此，8 名软件工程师聚集到一起制定了一套操作程序。这群人中的另一位成员兰迪・特布什说："我们决定使用手中现有的代码开始我们自己的项目。"[2]他们一致同意，要在明确的软件模块上设计并开发所谓的服务器程序，以便程序员能够轻松地在一个代码块上工作，而不必担心会影响整个程序。他们建立了一个简单的控制流程，只有在需求明确且得到其他成员同意的情况下，程序员才可以为其加入其他特性。他们将这一共同努力的成果称为阿帕奇（Apache），名字来源于最初被这些自嘲的开发者戏称为布满"补丁"的服务器。

自 1995 年年初，阿帕奇服务器软件就被一遍遍地重写，阿帕奇程序也不断发展成为拥有 40 万行代码的程序。一批又一批的程序员来来去去。竞争对手的程序都是由资深程序员编写、由实力雄厚的大公司推向市场；而免费共享、由世界各地的志愿者编码并调试的阿帕奇软件，却是市场上的领头羊。2001 年春天，有 60%的计算机使用阿帕奇软件提供互联网网页，将微软公司和 iPlanet 公司（由太阳微系统公司和网景公司创建的联盟，1999 年被美国在线收购）的产品远远甩在了身后。[3]阿帕奇的开发者承认，阿帕奇并不是最快的网络服务器，也不能提供最完善的特性服务，"但我们士气高涨"，布莱恩・贝伦多夫说。

阿帕奇是"开源软件"的成功典范之一。"开源软件"既代表一种哲学思想，也代表一种软件发展模式。在像阿帕奇和 Linux 这样的开源项目中，软件

是免费共享的，其“源代码”——经验丰富的程序员能够阅读并理解的编码指令——是公开发布的，以便其他程序员学习、分享和修改原作者的成果。开源模式是对现今大多数软件公司主导方式的一大挑战，因为在这些公司里，源代码被看成是代码提供者的私有财产。在计算机行业，软件通常都不是以源代码而是以二进制的形式（机器执行 1 和 0 的指令）分享流通的，普通人难以理解，尤其是当今的程序都包含数十万行甚至数百万行的指令。在严格的许可证条例规范下，软件公司有时也会与其重要的公司客户分享源代码，但这种分享也仅限于让大客户查看、理解代码，并不能像在开源模式里那样修改代码。

开源“运动”是理想主义、软件工程和商业战术的结合，20 世纪 90 年代互联网的迅猛发展使之成为可能。理想主义要归功于理查德・M. 斯托曼，他是 20 世纪 70 年代麻省理工学院人工智能实验室的明星程序员，后来创建了免费软件基金会。斯托曼认为，从自由出版研究的学术角度来说，所有的软件都应该是“免费的”，让其他学者来评论，与他人分享发现成果。对斯托曼来说，20 世纪 70 年代，当软件作为私人产品与硬件分开销售时，“闭源”软件的兴起是一个错误，是贪婪战胜了道德。

实际上，关于开源的“软件工程”争论在于，互联网作为低成本的即时全球交流媒介，已经从根本上改变了软件的发展模式。这一论证认为，相比于在某个公司单打独斗，互联网能够使程序员更加高效地与其他人分享想法、提出改进建议和修正程序错误。拥护开源的工程师认为，优秀的程序员将围绕他们感兴趣的软件难题，共同努力，相互激励，最终产生“精英代码”。他们断言，在互联网时代，多人的见解将有助于加快软件发展的脚步。这与软件工程学大师弗雷德里克・布鲁克斯的观点大相径庭。弗雷德里克・布鲁克斯曾指出，在一个大型项目中加入更多的程序员将会降低工作效率。埃里克・S. 雷蒙德是一位开源的拥护者，在 The Magic Cauldron[4]一文中，他宣称“分散合作的软件发展方式有力地颠覆了布鲁克斯的理论，将个人项目的可靠性和工程质量提高到一个前所未有的高度。”

开源同时也是一个商业策略，试图改变软件业界的交易条款，向占行业主导地位的供应商尤其是微软公司发起挑战，并分得一杯羹。为了实现这个目标，这些开源软件的倡导者全面革新并开展市场活动。

1998 年，他们用“开源”代替“免费”一词来形容他们的发展模式，因为“免费软件”似乎在暗示他们是反资本主义的极端主义者。而开源这一提法表明软件还是要挣钱的，只是软件会成为一种服务产业，而不是像传统模式那样，将软件程序看成是个人所有的制成品来买卖。服务费主要支付给帮助个人或者公司客户使用软件的人，而不是看着行业利益全部进了软件销售商的口袋。热心推广开源软件的人坚持认为，这种从销售方向用户方的权利转移才是公平的。他们说，毕竟只有 20%的专业程序员为软件销售商工作，而剩下的 80%都是各公司内部的程序员或者顾问，他们都致力于帮助人们使用软件。

开源这一提高软件效率的明智讯息一经发出就受到了广泛关注。很多大公司，尤其是 IBM 公司，已经支持并投资了两个主要的开源项目：阿帕奇和 Linux 系统。开源软件在一些前沿市场势头良好，如互联网服务器、好莱坞特技动画和超级计算领域等。欧洲将开源视作在软件领域赶超美国的大好机会。计算机专业的大学生认为开源很酷，因为它结合了编程的快乐、团队的友谊，以及支持外行和失败者带来的离经叛道的满足感。微软公司十分担忧；2001 年春天，微软派出其顶尖的公司高层人员向大学生和同行大力鼓吹“商业软件模式”的优点。开源软件能否成为主流还是个未知数，但它为软件界提供了另一种盈利模式。

布莱恩·贝伦多夫概述了开源运动面临的实际问题。确实，在讨论开源现象时，他避免使用“运动”这个词，因为它暗示了这是一种政治运动而非经济模式。“如果开源想取得成功，就必须有商业合法性，”贝伦多夫解释说，“很多人都加入了开源社区，因为他们认为这样做是正确的。但是慈善也只能做到

这里了，我们必须要让它可持续发展。”贝伦多夫头发齐肩，穿着袜子在Collab.Net 公司阁楼状的办公室里踱步。Collab.Net 是旧金山一家帮助其他公司设计并运行开源项目的公司。

贝伦多夫生于 1973 年，在南加州长大。他的父亲罗伯特和母亲贝基在 IBM 公司相识，他们都是 COBOL 时代 IBM 公司的系统工程师。贝伦多夫的家里满是个人计算机，他就是在这样的环境中成长起来的。四年级时，他开始了编程课程，学习 Logo 语言；Logo 是麻省理工学院的西摩尔·派珀特意为儿童开发的一门教育性编程语言。贝伦多夫后来尝试过 BASIC 编程，但是在六年级时，他“不再喜欢计算机了”，他说：“这是一种耻辱；计算机好像很神秘，但又没有什么意思。而且我觉得，孩子都有点叛逆，不愿从事父母从事的工作。”高中时代的贝伦多夫是个全能的学生，他参加越野跑，在学校的晚会上做节目主持人，还以接近全班第一的成绩毕业。他重新接触计算机是在进入加州大学伯克利分校之后，原因是他喜欢音乐。贝伦多夫沉迷于旧金山通宵的电子舞曲音乐会。在这里汇聚的主要是大学生，其中有些人接触到了 Unix 工作台和互联网，而贝伦多夫则很快就列出了“旧金山狂欢”参加者的互联网邮件发送清单，并建立了在线保存音乐文档。

1993 年，贝伦多夫得到了为旧金山一家新杂志社做“每小时 9 美元的 Unix 搬运工”的兼职工作。这家《连线》杂志社希望在互联网上创建电子邮件账户，并将文字版本的文章发布到网络上。早期，《连线》杂志用创新的彩色图片配以激情的文字在新闻界引领了思潮，它表示互联网不仅仅是一项技术，而且是一种社会文化力量。在 1993 年第一期《连线》杂志中，创建者路易斯·罗塞托宣称：“数字革命已经像孟加拉台风一样席卷了我们的生活。”这一讯息让贝伦多夫产生了共鸣。虽然只是个小时合同工，但他回忆说：“我觉得在《连线》杂志的一年半里，我好像在为上帝工作一样。”对于老一辈人来说，个人计算机是令人振奋的新技术，带来了社会变革的春风，是一种使个人拥有更多权力的技术工具。但是贝伦多夫这一代人将个人计算机视为理所当然，只有当它与

互联网连接、成为社交工具时才能体现其真正的意义。大多数加入阿帕奇项目的人都是如此。他说："我们中很多人都是随着互联网出生并长大的。"

阿帕奇项目开始时，贝伦多夫才 21 岁，那时他还继续做着《连线》杂志的工作。他们将伊利诺伊超级计算机中心的服务器编码作为工作的开始。"作为优秀的工程师，我们在编程过程中没有出现任何错误，"贝伦多夫说，"从第一天开始，我们就进行了多方努力。"要取得成功，开源项目必须有一个强有力的、团结的领导班子。阿帕奇项目做到了这一点，它的领导层是一组"红衣主教"，而非单一"教皇"；例如，Linux 系统的创造者林纳斯・托沃兹就扮演着教皇的角色。虽然关系很微妙，但这种集体统治至今仍在运作，这是因为阿帕奇小组成员之间有着共同的目标和价值观。他们的目标是建立一个能够处理不断提高流量的、高度可靠的网络服务器。阿帕奇项目小组由 Unix 程序员组成，他们用 Unix 的"工具"理念处理服务器设计，重点在于将每一块简单的软件积木精巧地结合在一起。简单性很容易实现，阿帕奇小组的兰迪・特布什说，这是因为他们没有商业压力。各种特性和修正都"出于需求，不会受赢得市场战争企图的驱使"。

服务器软件最初的设计并非出自阿帕奇小组，而是主要来自伊利诺伊超级计算机中心的罗伯・麦库尔，他离开中心后加入了网景公司。的确，程序创新基本上都是由孤独的手工艺人构思而成，顶多算是一些艺术家的设想。但是，正如开源社区的内部评论家埃里克・雷蒙德所做的巧妙形容，开源模式的真正力量在于将最初的设计放置在一个知识的"集市"中，程序员相互合作，不断改进、推敲和调试代码。

由于代码放置在公共域中，因此并没有从法律上禁止开源项目中某些人另辟蹊径。这就是所谓的"分流"；开源项目中，一项不可剥夺的基本权利就是"分流权"。从理论上说，这会让开源项目长久地处在分裂为不同组群的危险之

中，每一组群都可能发展成为一个程序版本，相互竞争。而实际上，两大旗舰项目并没有出现上述情况;阿帕奇和 Linux 系统都拥有极具说服力的有效领导。而这两个项目也因经济学中的“网络效应”团结在了一起；也就是说，使用的人越多，这种产品就变得越有价值。网络效应在软件程序这样的复杂技术产品中显得尤为明显。例如，有越多的人编写在阿帕奇和 Linux 系统上运行的应用程序，他们就“陷”得越深，因此就越不可能花费时间、精力和金钱在为其他服务器或操作系统重写软件上。这正是微软依托的经济学逻辑，微软依此将 Windows 操作系统打造成为了个人计算机行业中的技术标准。

两大阵营之间的不同之处在于，开源社区坚持认为技术标准应当公开，而不能仅为一家公司所有。开源的基本观点来自互联网及其在公共域公开标准的传统；从早期的通信及路由软件协议，如 TCP/IP，到网络上传送数据的基本标准 HTTP 皆是如此。开源社区认为不仅是通信协议，几乎每个人都在使用的“框架”软件都应当是免费公开的。HTTP 网络服务器之一的阿帕奇就是这样的一个程序，他们坚持认为操作系统也应当如此，因此有了 Linux 系统。开源社区的这种立场无疑与业界龙头老大微软公司唱起了反调。微软公司将其 Windows 操作系统从个人台式计算机引入业界具有优势的大型服务器计算机之中，这些服务器计算机用来支持公司数据系统和互联网商务等各种业务。但是开源社区认为互联网技术、经济和命运都站在他们这边。“这么说吧，”贝伦多夫说，“我认为微软公司作为一家操作系统公司，前途不会一片光明。”

1994 年，贝伦多夫在大学三年级时从加州大学伯克利分校退学，全身心投入到互联网计算和开源软件当中。他说：“我从没有后悔。”1995 年，离开《连线》杂志之后，贝伦多夫创建了一家名为 Organic Online 的网络设计咨询公司，并担任首席技术官。1999 年，他又帮助别人创立了 Collab.Net 公司，同样担任首席技术官。

贝伦多夫是个实用主义者，在开源的经济与商业合理性方面如数家珍，但同时他也相信软件不单单是一种重要的商业工具，其重要性还体现在社会和政

治价值上。他将开源运动与美国独立战争进行了大致的比较。“它们都与控制和分权控制有关，”他解释说，“编程是一种权力，因为代码执行的是创作者的方针政策。在编程环境中，你的知识越多，就越有权力。”

像许多人一样，贝伦多夫也看到了在日益普及的数字网络中，软件在商业、通信和教育领域作为新兴技术的未来。他相信应该有一个民主的社会目标，每个人都能或多或少地控制软件，而不是为软件所控制——能够控制软件环境为隐私、安全、娱乐和其他事物服务。也就是说，能够编程。“至少每个人都应该学会编程，”贝伦多夫说，“这并不是意味着你要学习 C 或者 C++语言，但是不会编程就好像不会驾驶，缺少了一项基本的社会技能。”

手工艺人通常对自己使用的工具有一种特别的喜爱。一个工作台和一套工具，这样一个常见的、设备齐全的环境就足以令人满意并实现梦想了。如果程序员与他的计算机和软件变得密不可分，那么这种联结的力量就尤为强大了。心爱的机器、编程语言或者操作系统能够塑造程序员的观点和心智。例如，每一种语言和操作系统都有其优缺点，它们在为程序员提供自由的同时，也对其加以限制；这一系列规则就是一种技术哲学，一种工作文化。在一些极端的例子当中，手工艺人和工具之间的联结关系更加深厚，即使不是一种信仰，也几乎是一种生活方式了。没有比在理查德・M. 斯托曼身上所发生的转变更惊人的例子了。故事开始于 20 世纪 70 年代，他在麻省理工学院富有特色的人工智能实验室里，负责研究美国数字设备公司 PDP-10 微计算机的操作系统。

斯托曼生长在纽约，是个数学神童，8 岁时就开始涉猎微积分了。几年之后，他的一个夏令营辅导员碰巧拿到了一份 IBM 7094 大型计算机的使用手册，斯托曼看到后就阅读起来。他开始编写一些简单的程序，如分别算出一组数字的立方数，再将这些得数相加。他没有机会接触计算机，但是斯托曼回忆说：“我就是想编写程序，我对它很着迷。”[5]上高中的时候，他参加了一个能让天

才学生接触 IBM 大型计算机的课外项目。由于斯托曼操作熟练，IBM 在他进入哈佛大学前雇用了他工作一个暑假。在哈佛大学，他有机会使用到应用数学系的分时计算机。由于资源有限，在计算机上并不能进行永久存储，因此当用户退出分时系统计算机时，其工作成果就会被删除。但是斯托曼很快就发现，如果他同时在两个终端上登录，他的编程文档就不会被删除了。他玩这个小把戏只是觉得好玩，只是证明他能够做到，事实上他从没有私自占用过存储空间和终端时间。“我不喜欢规则，”他回忆说，“现在也是一样。这只是一种表达我对规则看法的智力练习。”腼腆、害羞却非常理智的斯托曼，在软件中找到了最适合表达自己和创造事物的媒介。“唯一能让我进行创造的东西就是软件。有了软件，就没有现实事物中所受到的约束了。这就好像你可以四两拨千斤，完成现实中不可能做到的事。”

1971 年的一天，在斯托曼快要结束第一学年的时候，他穿过马萨诸塞州的剑桥，来到了麻省理工学院的人工智能实验室。离开时，他已经得到了一份系统程序员的暑期工作。1974 年，斯托曼从哈佛大学毕业，取得了物理学学士学位；不过，他的热情一直都是在麻省理工学院的实验室里编码，直到 1984 年离开。在人工智能实验室的 13 年中，斯托曼几乎没有为人工智能编写过程序，他的工作重心是实验室中 PDP-10 机器的操作系统。在 20 世纪 70 年代初，麻省理工学院人工智能实验室是个学术自由、技术上持乐观态度、反对正统文化的地方。这里的环境公开透明，一切都是自由分享的，没有什么是私有的，不存在任何等级和特权。斯托曼回忆说，研究操作系统就好像是在一个公共的软件沙盒里玩。“我们欢迎任何有能力的人来一试身手，改进操作系统。我们在彼此的工作成果上工作……他们放下的工作，我拿起来就可以继续。”

在实验室里，斯托曼还深入熟悉了人工智能语言 Lisp。对斯托曼来说，Lisp 是一种理想的编程语言，它的设计富于想象力，也没有其他语言具有限制性的规定。Lisp 语言避开了其他语言所具有的数据定义的层级，其简单的句法又拉近了数据与编程指令间的距离。甚至时至今日，斯托曼谈起 Lisp 强大的吸引

力时，还是会满腔热情地说起“它的高雅，它简单的句法，它的灵活性和它的力量”。

程序员在麻省理工学院实验室里拥有权力，这激励了斯托曼并对他产生了一些政治化的影响。“我很胆小，不敢向权威提出挑战，”他说，“但是人工智能实验室教会了我这样做。”来到实验室后不久，斯托曼注意到一个安装在轮子上的钢筒。曾经有一位教授将他的计算机终端锁在自己的办公室里，大家就使用这个钢筒撞开了他的房门。斯托曼解释说：“任何一个有计算机终端却不使用它的人都必须放弃它。”程序员的规定约束着每一个人，甚至是试图私自占用终端的麻省理工学院教授。“人工智能实验室的计算机黑客不允许那样做。”他说。

但是到了 1981 年，麻省理工学院实验室开始发生了变化，人们纷纷离去创建公司。他们大量借用麻省理工学院实验室的工作成果，生产专门的 Lisp 语言机器来谋求财富。但是这些公司都为他们的软件加上了所有权，互相竞争，拒绝分享。他的老朋友们都离开了实验室。在斯托曼看来，实验室特有的文化已经被目光短浅的商业贪欲毁于一旦。1984 年 1 月，斯托曼从麻省理工学院辞职，开始追求他狂想家式的、解放软件世界的使命。“我得出的结论是，软件私有是错误的，”他说，“这是要将人们分割开来，使他们陷入无助的境地。我决定要与之战斗，结束这一切。”

2000 年冬季的一天，长发披肩、蓄着连鬓胡子、绿色眼睛神采奕奕的斯托曼坐在位于纽约市他母亲家的厨房里，一边整理着缠结在一起的头发，一边在一台小小的银色笔记本计算机上敲着 Lisp 编码，一边回答问题。他是一个怪人，却有着奇特的魅力和自嘲的幽默感。斯托曼是出版于 1983 年的《黑客词典》的撰写人之一，在词典里斯托曼对自己的描述是：“我是 1953 年左右在曼哈顿的一间实验室里被创造出来的，并于 1971 年移居麻省理工学院的人工智能实

验室……大约一年前，我与‘结婚’10 年的 PDP-10 分手了。我们仍然相爱，但是世界将我们带往了不同的方向。”朋友和同事都用他名字的首字母 RMS 称呼他，以表达对他在麻省理工学院登录计算机方式的敬意，斯托曼自己也欣然接受。“理查德 · 斯托曼只是我在尘世中的名字。”他于 1983 年写道。

斯托曼于 1984 年离开麻省理工学院，投身到反对软件私有体制的战争当中，他仅有的武器就是编写软件并免费共享。他选择的临时目标是 AT&T 公司的 Unix 操作系统。斯托曼将他的项目称为 GNU，这是 Gnu’s Not Unix（Gnu 不是 Unix）有趣的递归首字母缩写。斯托曼是公认的伟大程序员。1990 年，由于开发了成为资深程序员遵循标准的编辑程序 Emacs，他作为“杰出青年计算机专家”，赢得了美国计算机学会设立的年度葛丽丝 · 霍普奖。一个像 Unix 这样已经成型的操作系统包含很多内容——一系列开发工具、程序库和编译器等。1984 年，斯托曼开始了试图在每个领域都赶超 Unix 的浩大工程。他的工作成果将与全世界分享，同时邀请其他人加入开发行列。几年过去了，斯托曼和一些合作者在开发与 Unix 兼容的操作系统上取得了惊人的进步。GNU 编译器和系统程序库都十分卓越，GNU 甚至还含有一个国际象棋游戏。毕竟，Unix 带有一个国际象棋游戏，那么 GNU 也要有。斯托曼解释说：“下国际象棋，这是一台计算机应该做到的。”

但是斯托曼的工作并未触及设计操作系统的“内核”——程序的核心，它最接近计算机，控制着大部分基本操作。在 1991 年的芬兰，赫尔辛基大学的学生林纳斯 · 托沃兹开始着手研究内核问题。他参加了学校第一期“C 语言和 Unix”的课程，被 Unix 的理念及其产生的“无瑕疵的漂亮操作系统”[6]所折服。托沃兹由此产生了灵感，他通过分期付款购买了一台价值 3000 美元的个人计算机，并且最终编写出了自己的 Unix 型内核。托沃兹将他的工作成果放到互联网上，招募感兴趣的程序员给他帮忙。这些程序员在网络上聚集起来，人数一直在缓慢地增加。托沃兹编写的代码给他们留下了深刻的印象，托沃兹本人也成为非正式却毋庸置疑的项目领导者。这些程序员甚至还举行了捐款，帮托

沃兹购买了他计划分 3 年付款买下的计算机。他们提出建议，不断精炼和改进内核。一个工艺精湛的内核固然至关重要，但也只是操作系统的一个组成部分。对于剩下的部分，托沃兹和他的追随者们使用了现成的免费软件，其中包括 GNU 的编译器和 BSD 项目开发的软件；BSD 是 1979 年起加州大学伯克利分校开发的 Unix 变种软件。他们最终完成的操作系统被称作 Linux，是“林纳斯的 Unix”的缩写。

Linux 项目的出现正好赶上两大技术潮流——功能日益强大的廉价个人计算机兴起和互联网大爆炸。一夜之间，廉价的机器就可以运行相当复杂的软件，互联网也方便了野心勃勃的程序员之间的交流与合作。开源运动评论家埃里克·雷蒙德评价托沃兹说，他并不是一个能与斯托曼和 Java 语言之父詹姆斯·高斯林并驾齐驱的软件设计魔法师，但是他的天赋在于创造出软件发展的新模式。雷蒙德写道，托沃兹是“第一个学会适应互联网普及后的新规则的人”[7]。当被问起这一点时，托沃兹回答说：“我真正的天赋是很好地将技术与交流结合起来。”[8]之后，他沉思了一下又说：“是什么使 Linux 系统与众不同？也许是因为我对政治不太感兴趣。”托沃兹确实牢牢控制着 Linux 编码的开发原则，认为所有操作系统的发展进步都要放入项目中，进入公共域。“因此我让人们改进 Linux 系统，”他说，“而不被免费软件基金会试图推行的愚蠢政治议程搞得心烦意乱。”

在林纳斯·托沃兹看来很愚蠢的问题，对于理查德·斯托曼而言却至关重要。有很多事情都激怒了斯托曼，但是令他尤为气愤的是，人们经常误以为他是开源社区的一员。他怒言道：“这就好像把富兰克林·罗斯福称为一个共和党人。”开源信息主要提高了效率。对用户而言，这种模式会产生更好的软件，尽管一些软件供应商的利益会因此受到伤害。而斯托曼的自由软件运动强调的是道德性。“开源软件的主要目的不是自由，而是胜利，”斯托曼说，“这是多

么肤浅又没有意义的目标啊！”斯托曼用话语发动了这场运动，带着道德家对自身正义性的肯定和工程师对精准的坚持。因此，在他的定义中，免费软件和开源软件是两个不同的概念，虽然它们开展的方式大同小异。

同样，斯托曼坚持说 Linux 恰当的名字应该是 GNU Linux，因为免费软件基金会的 GNU 项目为它做了许多贡献。他说，将它简单地称为 Linux 是“不公平和不公正的”。作为回应，托沃兹说他并不在意操作系统叫什么名字，还说斯托曼关于 GNU Linux 的争论“一直都没有什么说服力，因为大部分代码来源于 BSD”，也就是加州大学伯克利分校软件，以及其他开源项目，而不是免费软件基金会。相对于代码，意识形态的分歧再一次激怒了斯托曼。“有数百万的用户使用该系统，却对其背后的价值体系一无所知。”他解释说，并失望地表示，大部分 Linux 程序员认为 GNU 的贡献仅仅是少数几个工具。开源软件的用户“只感激我们种下的几棵树，而对我们建造了整个森林的事实却只字不提。”

斯托曼多年来的自由软件运动促使许多人重新思考软件是如何生产出来的，重新考虑将代码看做知识产权、为某个公司所专有的隐含意义。但是自由分享软件得以在数百万计算机上运行，要归功于林纳斯 · 托沃兹这样的开源温和主义者。托沃兹是位芬兰人，一头金色头发，胡子刮得干干净净，现在居住在美国硅谷。对很多人来说，托沃兹是一个大众可接受的计算机黑客形象——注重实际、自谦、广受欢迎，就像他为 Linux 系统选择的吉祥物——惹人喜爱、胖胖的卡通企鹅。当然，这部分是出于市场策略——一个现代的“忠厚老实人”在迎合他的观众。他在技术问题上也可能会像斯托曼那样斤斤计较，独断专行。事实上，托沃兹曾说他讨厌 GNU Emacs 编辑器。尽管很多人将它视为富有创造力和“激情”[9]的艺术杰作，但托沃兹认为它“糟糕透顶”，对它不屑一顾。不过，托沃兹是一位颇具魅力的开源软件大使。斯托曼靠麦克阿瑟基金会的投资收益和演讲收入度日，公司客户通常会排斥他过于坚持的意识形态，认为他有胁迫性，令人不快。而托沃兹是商业社会中的正式成员，为全美达公司工作，

为微芯片制造商编写软件。

托沃兹承认，他不像斯托曼和免费软件基金会里的其他人那样，具有“麻省理工学院激进主义背景”。托沃兹说：“RMS 像是佛教徒，他牺牲自我以唤醒民众。他坚信开源软件的必要性，这很好。但这使他成为一个在正常情况下极难相处的人，除非把他当做一个偶像。这就使许多人真的很不喜欢他和他所代表的一切。他太顽固、太虔诚了。我的观点是，开源一旦脱离了免费软件基金会的政治理念和价值观会走得更好，而且会有更多的人觉得开源软件是工具而不是宗教。我是个绝对的实用主义者。”

受实际利益的驱使，各大公司也纷纷开始支持开源软件，IBM 公司就是最典型的代表。这家世界最大的计算机公司在其自有网络服务器软件失败的情况下，于 1998 年开始支持阿帕奇服务器软件。但是真正的重大举措是在 2000 年，IBM 宣布将支持 Linux 系统。这是 IBM 公司试图将操作系统转变成无收益商品的一项策略，用以打击两个主要对手——微软和太阳微系统公司。在大型计算机之外的现代操作系统市场中，IBM 是个失败者。它在个人计算机软件领域的成就 OS/2 及其 Unix 风格的操作系统 AIX，被微软的 Windows 和太阳公司的 Solaris 操作系统轻而易举地击败了。因此，根据长久以来的逻辑“敌人的敌人就是朋友”，对 IBM 而言，Linux 这个看起来矮矮胖胖的小企鹅也变得可爱了。

而更重要的是，IBM 开始相信 Linux 及其开源发展模式将成为赢家。IBM 的主要业务资产是与公司客户的长期关系。如果 IBM 出于竞争的考虑，在技术上误导了这些公司客户，损害了它们的利益，那么它们最终将会抛弃蓝色巨人（Big Blue）[①]。所以 IBM 非常看重 Linux 和开源。IBM 押宝 Linux 是最高管理层作出的决策，但这一策略的形成有着更广泛的群众基础。其中一位关键人

① 蓝色巨人（Big Blue）指的是 IBM。蓝色是 IBM 公司用于公司标志的识别颜色，蓝色巨人的名字由此而来。——译者注

物就是物理学博士埃尔文·瓦拉达斯基·博格[10]，他是公司最高管理层信赖的技术顾问，长期以来都是 IBM 业务和研发领域的元老。他虽然挂着公司副总裁的头衔，却并不实际参与公司管理。2000 年 1 月，在 IBM 作出决定之前，他亲自向董事长小路易斯·V. 郭士纳及其他高层管理人员做了简短的报告，核心要义是 Linux 和开源软件可能成为“又一项突破性的技术”，就像 20 世纪 80 年代 IBM 紧跟的微处理器潮流，以及 20 世纪 90 年代 IBM 迅速吸收的互联网技术。“虽然我还无法说出它确切的走向，”他对他们说，“但是我感觉这又像是一个大事件。”

在前两次突破性技术浪潮中，瓦拉达斯基·博格直接参与推动了 IBM 的战略调整。20 世纪 80 年代，他在担任大型机发展部门副总裁时就发出警告，低成本的微型处理器技术将对 IBM 大型机造成冲击。不但较小的机器将运行微处理器，接手先前由 IBM 巨型机处理的琐事，就连大型机本身也不可避免地要采用大规模生产的微处理器技术，替代过去本质上依靠手工制作的双处理器。不过瓦拉达斯基·博格承认，就连他自己也没有预见到微处理器革命的到来如此迅猛，给 IBM 造成了巨大的经济损失和管理动荡。20 世纪 90 年代，当互联网腾飞的时候，瓦拉达斯基·博格被任命为互联网部门的总经理，他的工作是确保公司不会再次被打个措手不及。他扮演了传教士和管家的角色，要确保 IBM 的各项业务与公司的互联网战略协调一致。在 20 世纪 90 年代，IBM 将自身定位成了为各类公司客户提供互联网技术和经验的供应商，这一敏锐的定位促进了公司的兴旺发展。蓝色巨人吸取了教训：先下手为强，开发未来的技术资源，不能落后、被过去的过失所拖累。

埃尔文·瓦拉达斯基·博格是东欧犹太人移民的后代，在古巴出生长大。他说，他的复姓是为了“听起来像古巴人，虽然现在看来有些奇怪”。他的父亲朱利叶斯是俄国人，在哈瓦那经营一家廉价物品专营商店。1959 年，菲德

尔·卡斯特罗当权后，他父亲变得一无所有了。于是全家人在 1960 年来到美国。虽然行囊空空，但年少的瓦拉达斯基·博格还是从古巴带来了毕生热爱的棒球。他是纽约大都会队的球迷，对 1982 年纽约洋基队老板乔治·史坦布伦纳不再续签雷吉·杰克逊耿耿于怀。得益于在古巴的成长经历，他爱好拉丁爵士乐，如庞秋·圣骑士、狄托·庞特、帕奎托·D. 里维拉等，还酷爱埃尔莫·伦纳德的小说。“我喜欢看迈阿密下层社会的场景。”他说。

来到美国之后，瓦拉达斯基·博格一家定居在芝加哥，因为他们在那里有一些亲戚。瓦拉达斯基·博格很快就学会了英语，并成为了优等生。1962 年的夏天，瓦拉达斯基·博格在去芝加哥大学上大一之前，在芝加哥大学新建的计算机中心得到了一份工作，并一直做到大学毕业。他早期编写的程序主要是运用 IBM7090 大型计算机帮助教授解决一些科学问题。例如，他曾帮助一位生物学教授编写数学模型程序，以测定在电击时鱿鱼神经细胞和神经轴突的反应情况。他的大部分程序是用一种叫做 FAP 的汇编语言编写而成的，并通过“八进制输出文件内容”进行调试，这是一种基于八进制计数法的计算机内存输出文件。

“编程真的很有意思，”瓦拉达斯基·博格回忆说，“让计算机做你想让它做的事，让复杂的程序运作，这真的可以给人成就感。你能感觉到一种令人难以置信的力量。”他觉得调试修正程序好像是“侦探工作”，因为必须“准确搜索到问题到底出在哪里”。他经常熬夜运行有问题的程序，先提交代码，然后在午夜时分到 Unos or Dues（现在在芝加哥还很有名的比萨店），或者城南的墨西哥餐厅 Taquerias 吃夜宵，凌晨两三点再回去。“希望代码已经运行完毕，”瓦拉达斯基·博格回忆说，“在 19、20 岁时，这真是件很酷的事，不过我猜这会让人觉得我是个书呆子。”

瓦拉达斯基·博格身材不高，戴眼镜，有着卷曲的灰色头发，是 IBM 公司的书呆子。1999 年，无论是从超级计算机领域研究员、互联网程序员，还是从来自美国白宫信息技术顾问小组（由世界领先的计算机科学家组成）的朋友

口中，瓦拉达斯基·博格都越来越多地听说 Linux 系统。IBM 公司已经开始支持阿帕奇软件，而且它还有一个规模不大但不断成长的程序员小组在参与包括 Linux 在内的开源软件项目。10 月底，公司的高级副总裁萨缪尔·帕米沙诺结束全球旅行，回到了公司。他从访问的公司得知，年轻的互联网程序员都喜欢 Linux 操作系统。后来成为 IBM 总裁的帕米沙诺立即开始了"公司评估"，研究 IBM 公司对于 Linux 的应对策略。

1999 年 10 月 30 日，一个星期六的下午，IBM 公司的高级研究员尼克·鲍文在康涅狄格州纽敦市的家中接到了一通电话。电话是他的老板、沃森实验室负责人保罗·霍恩打来的。鲍文的任务是领导一个 11 人的团队，研究 Linux 系统的机遇与风险并给出建议。鲍文在 12 月 20 日向帕米沙诺提交了报告。这份计划书旨在削弱微软公司和太阳微系统公司在数据服务器计算机操作系统中的优势，目的是赢得计算机领域最有影响力受众的心——他们就是编写应用软件、将网络带给千家万户并使促成电子商务的软件开发者们。这份报告称："今天，微软公司和太阳微系统公司主导着应用软件的发展。我们建议 IBM 公司大举研究基于 Linux 系统的应用软件开发平台，并且支持其全线产品，这样才能打破太阳和微软的束缚。"报告还指出，Linux 策略还将帮助 IBM 公司树立起站在技术前沿的形象，使之能够在"软件发展的关键领域和大学中"与太阳微系统公司和微软公司一决高下。[11]

帕米沙诺很喜欢这份报告。与郭士纳交换意见之后，他于 2000 年 1 月 7 日向公司的高级管理层发送了一份电子邮件备忘录，宣布"我们将支持 Linux 系统"。瓦拉达斯基·博格被任命为 IBM 公司 Linux 项目的负责人，并继续融入开源社区之中。瓦拉达斯基·博格低调、专业的工作作风和深厚的技术背景使这个能说会道的 IBM 书呆子似乎是公司支持软件反主流文化、使公司计算机人性化的理想宣传大使。瓦拉达斯基·博格指出："Linux 不仅仅是一个操作系统，更是一种文化。"

IBM 等公司并不是唯一在开源软件的巨大潜力中寻求突破的组织。欧盟委员会的一个技术咨询小组也曾将开源软件称为欧洲地区的“巨大机遇和重要资源”[12]。这个小组在 2000 年一份名为《免费软件/开源：欧洲在信息社会中的机遇？》的报告中指出：“欧洲的公司和开发者已经是许多开源项目的驱动力量。如果开源软件能够改变信息技术产业的规则，那么更加理解并善于运用开源软件的公司与国家将具有明显的竞争优势。”欧盟企业与信息社会委员埃里奇·利卡宁说，聚焦像 Linux 这样的开源软件项目，“也许能激发欧洲人在这一领域的创造力，并显著降低我们对于进口的依赖程度”。

开源软件能走多远还是个未知数。只要开源软件试图在传统商业软件公司主导的市场环境中求生，它就注定会举步维艰。2001 年 5 月，由原苹果公司 Macintosh 团队成员组建的开源软件公司 Eazel，由于不能说服投资者继续支持这种尚未成熟的商业模式而关门停业。Eazel 公司的创建者之一、为 Linux 设计了图形用户界面的安迪·赫兹菲尔德对此感到很难过，但仍对开源软件充满了热情。“开源软件对用户和开发者来说是一件好事，因此，它终将成功，”他说，“这是迟早的事。”[13]虽然赫兹菲尔德离开了 Eazel 公司，但他为开源软件的发展模式所折服。在开源软件这个“自成一体的精英制度”中，“精华会浮到顶部，一流的技术决定通常都来自混沌之中”。但是有一点令他不以为然，这就是他所谓的“过于狂妄的观点，因为开源软件显然不是开发伟大软件的唯一出路”。

早在 20 世纪 70 年代末、80 年代初，来自加州大学伯克利分校的比尔·乔伊在“开源”这个词产生之前，就开启了开源软件的风潮。他是伯克利 Unix，即 BSD 的主要设计者，并将该软件免费分享。“在我心里，BSD 是一项研究，”他回忆说，“如果它是一项研究，那么就应当公布出来。”伯克利 Unix 这一创举更多是出于大学和研究室的需要。乔伊启用了互联网初期的通信协议

TCP/IP，将它们整合到一起，并修正了错误，以达到更优性能，使之趋于完美。在伯克利 Unix 中，他建立了互联网的公开协议，作为大学网络标准，而不是像美国数据设备公司、IBM 公司和其他公司那样将软件私有化。

乔伊对于开源运动经历了一个从困惑到赞成，再到拒绝的转变过程。[14] 乔伊说，互联网和廉价个人计算机的普及已经改变了环境，合作和调试软件变得更容易了。“但是真理在于，伟大的软件来自于伟大的程序员，而不是劳苦工作的大众。”他于 1982 年建立了太阳微系统公司。他说：“开源软件大有裨益。它能够加快事情发展的步伐。但是它已经不再是新技术，也不是灵丹妙药。”对于抛弃将程序员视为手工艺大师或艺术家的商业模式，乔伊持怀疑态度。“创造者得到的金钱是对他创造力的一种奖励，”乔伊说，“我觉得说那种商业模式已经结束还为时过早。它现在仍然运作良好。”

根据过去的经验，对于创建程序的工具师来说，开源将是另一个新秀。它能帮助程序员开发出更好的软件，并且提高速度。它是手工艺人手中一个功能强大的工具，但并不是万能的魔杖。C++语言的创造者、学习哲学的丹麦人本贾尼·斯特劳斯特卢普指出：“开源软件是个好想法，然而，试图把它当成唯一的办法永远都是一个错误。”[15]

即使这个世界一点一点地变得越来越软件化，但它永远不会是二进制的。

后　　记

程序员经常使用工具（tool）这个词描述他们开发的软件以及他们应用的编程工具。工具这一说法采用了手艺人和工程师的语言和视角。程序员的工作领域是“计算机科学”，但这个词可能不太恰当。软件工程专家弗雷德·布鲁克斯指出：“它是工程原则，而非科学原则。”这两种视角截然不同。布鲁克斯解释说，科学家构建的目的是研究，而工程师研究的目的是构建。

很多领先的程序员原先都从事科学或数学领域的工作，在发现软件的实践工具比其他领域的理论更有吸引力之后，转而进入软件行业。巴特勒·兰普森讲述了这一转变过程；他在施乐 PARC 的研究奠定了现代个人计算机的基础。20 世纪 60 年代，兰普森在加州伯克利大学攻读物理学博士学位。一天，他走进了学校的计算机实验室，后来就再也没有回到物理专业。他回忆说：“我在那里徘徊，被计算机迷住了。”像其他很多转行到计算机领域的物理学家和数

学家一样，兰普森常常谈到工程学带给他的满足感——用软件创造事物，既不需要耗费大量体力，也不需要卡车和起重机吊装，唯一需要的就是将知识组合在一起。兰普森说，编程是一种自由的工程。“工程师热衷于创造事物。对于任何迷恋计算机的人而言，这就像是毒品。”

就在本书将要完成的时候，计算机领域又有了另一项革命性进步，这就是所谓的“栅格”计算。这预示着有朝一日智能计算机的力量将为人所用，随时随地为人们提供所需的帮助——只需插入或无线接入，就可以获得超级计算机的能力。栅格的概念在很多方面都为人所熟知，这也是对 J. C. R. 利克里德关于“人机共生”以及追溯到 20 世纪 60 年代信息效用理念的回应。就像从分时系统到互联网等一系列创新一样，栅格计算也是由大学和政府的实验室率先开始研究的。

和计算机发展中的其他重要进步一样，栅格概念的实现得益于处理能力、网络容量以及软件等方面的持续进步。按照最宏伟的构想，栅格将成为解决人类一切问题——从疾病治疗到游戏娱乐——的技术基础。虽然栅格现在还没有实现，但数十年之后，社会大众可能会利用计算机来提高人类智能。

栅格如果能实现的话，将极大地依赖于软件。在某种程度上，栅格软件被视为对 Java 等的巨大跨越，能够有效地推动互联网的智能化和可编程化。作为位于芝加哥附近的阿尔贡国家实验室的资深科学家，新西兰籍的依安・福斯特现为栅格软件设计的带头人之一。福斯特和他的同事（通常都是这样）南加州大学的卡尔・凯瑟曼在几年前就开始着手一个编程问题，并且工作不断扩展。“我们正在构建的工具有广泛的用途。”福斯特回忆说。在软件领域，从 FORTRAN 到 Unix 再到 Web，类似的故事比比皆是。工具越来越有用，程序员的视野也在不断扩展。

编程的创新来自工程的多样性，部分是实用发明，部分是结构设计——就像托马斯・爱迪生和弗兰克・劳埃德・赖特。在软件领域，突破性的进展不同于科学和数学领域的基础理论见解，例如顿悟 $E=mC^2$。相反，自 FORTRAN 之

后，编程创新指的是发现一种创造新事物的方法，或者用一种新的方法创造事物，然后再去实践。在这种情况下，软件大师更像建筑师或土木工程师，即用不同的方式进行设计和建造。如今的大型软件程序比人类建造的任何其他事物都更加复杂。

最好的程序员既擅长概念性思考，又善于把握程序细节；从高级语言设计到贴近机器底层的实施，转换自如。对于那些擅长此道的人而言，运用这种能力并不全是出于乐趣，有时也是莫名地情不自禁。虽然这一点很难向非业内人士解释清楚，但《计算机程序设计艺术》一书的作者，斯坦福大学的高德纳就作出了很好的说明。高德纳解释说，他在 60 多岁的时候还觉得每天都需要编写程序。“编程具有一种独特的美，所以我必须这样做，”他说，“我喜欢看它们组合在一起为你‘弹奏’每一个音符。”

注　　解

第 1 章

[1] 2000 年 10 月 10 日，作者访谈。

[2] 1999 年 11 月 18 日，作者访谈。

[3] 马丁·坎贝尔–凯利和威廉·埃斯普瑞，*Computer: A History of the Information Machine*，第 181 页。

[4] 波斯数学家。高德纳，《计算机程序设计艺术，卷 1：基本算法》，第 1-2 页。

[5] 2000 年 9 月 11 日，访谈。

[6] 摘自 1995 年 2 月 10 日出版的珍·巴蒂克 21 页的个人史，第 9 页。

[7] 克里斯托弗·埃文斯 1976 在加州对葛丽丝·霍普的访谈，现为查尔斯·巴贝奇研究所口述历史项目的一部分，印刷本，第 8 页。

[8] 2001 年 1 月 5 日，访谈。

[9] 莫里斯·威尔克斯，*Memoirs of a Computer Pioneer*，第 145 页。

[10] 首次公开发表。该发现是耶鲁法学院的图书管理员弗雷德·夏皮罗做出的。“软件一词的来源：JSTOR 电子期刊档案中的证据”，*IEEE Annals of the History of Computing*, 22(2000)：第 69-71 页。

[11] *IEEE Annals of the History of Computing*，第 6 卷，第一期（1984 年 1 月），第 16-17 页。

[12] 2000 年 9 月 11 日，访谈。

[13] 斯蒂芬·亨德里克和卢多维克·布鲁诺，*The 2001 IDC Professional Developer Model*，2001 年 6 月出版。2000 年，国际数据公司的数字为 873.9 万，预计 2011 年增为 978.3 万。

[14] 总统信息技术咨询委员会 1999 年 2 月 23 日发布的 *Information Technology Research: Investing in Our Future*。

[15] 引自高德纳的《计算机程序设计艺术，卷 1：基本算法》，第 231 页。

[16] 2000 年 6 月 19 日，访谈。

[17] 2011 年出版了第 4 卷的第一部分 4A 卷。——编者注

[18] 2000 年 12 月 7 日，访谈。

第 2 章

[1] 查尔斯·拜什等人，*IBM's Early Computers*，第 130-135 页。

[2] 出自 1982 年 5 月 26 日 IBM 的 FORTRAN 纪录片，13 分钟。

[3] 若无其他注释，则全部引自巴克斯，大部分对他背景及作品的描述源自四次访谈。其中，两次面谈分别于 2000 年 8 月 7 日和 9 月 29 日在他旧金山的家中进行，另两次是 2001 年 4 月 27 日和 5 月 3 日的电话采访。

[4] IBM 的部分资料显示是 11 个人，包括葛丽丝·米切尔。依据巴克斯和齐勒的说法，葛丽丝致力于参考手册以及向用户传播 FORTRAN I。

[5] 2000 年 9 月 12 日，访谈。

[6] “程序语言，过去、现在和未来”，《IEEE 展望未来》，学生通讯，1996 年秋。

[7] 2000 年 9 月 22 日，访谈。

[8] 2000 年 9 月 11 日，写给作者的邮件。

[9] 2000 年 8 月 7 日，访谈。

[10] *IBM's Early Computers*，第 162 页。

[11] 2000 年 8 月 7 日，访谈。

[12] 若无其他注释，则全部引自欧文·齐勒的三次访谈：2000 年 9 月 12 日，11 月 2 日和 2001 年 4 月 27 日。

[13] 1982 年 IBM 文件记录。

[14] 2000 年 9 月 18 日，访谈。

[15] 2001 年 3 月 13 日，访谈，丹・麦克拉肯讲述。

[16] 1987 年 5 月 14 日，威廉・埃斯普瑞对维勒的访谈，作为查尔斯・巴贝奇研究所口述历史项目的一部分，第 3 页。

[17] 葛丽丝・霍普访谈，查尔斯・巴贝奇研究所口述历史项目，第 14 页。

[18] 保罗・克鲁兹，*A History of Modern Computing*，第 85 页。

[19] 坎贝尔–凯利和埃斯普瑞，*Computer*，第 187 页；FORTRAN 团队访谈。

[20] 克鲁兹，第 86 页。

[21] *The History of Fortran I, II and III*，1979 年首次出版，再版于 *IEEE Annals of the History of Computing*，第 4 期，第 20 卷（1998 年），第 70 页。

[22] 《ACM 通讯》第 3 期第 11 卷（1968 年 3 月），第 147-148 页。

[23] *Preliminary Report: Specifications for the IBM Mathematival GORmula TRANslating System*，11/10/54，第 2 页。

[24] *History of Fortran I, II and III*，第 72 页。

[25] IBM 纪录片，1982 年。

[26] 2001 年 5 月 3 日，访谈。

[27] *History of Fortran I, II and III*，第 74 页。

[28] 通过一个例子来看，丹・麦克拉肯提供的 Fortran 编程范例很有帮助，然后由 Lahey 计算机系统公司提供的 FORTRAN 编译器将其转换成汇编语言，并生成二进制。

[29] IBM 纪录片，1982 年。

[30] 2001 年 4 月 28 日，访谈。

[31] 2000 年 11 月 2 日，访谈。

[32] 2000 年 9 月 12 日，齐勒访谈。

[33] 2000 年 9 月 12 日，访谈。

[34] 2001 年 4 月 27 日，访谈。

[35] 2000 年 12 月 6 日，访谈。

第 3 章

[1] 2000 年 9 月 18 日，作者访谈。

[2] 除另有注释出自 2000 年 12 月 6 日在斯坦福的访谈，则引自约翰·麦卡锡。

[3] 1989 年 4 月 18 日，1990 年 11 月 14 日，亚瑟·诺伯格对考巴托的访谈，作为查尔斯·巴贝奇研究所口述历史项目的一部分，第 12 页。

[4] 1959 年 1 月，见约翰·麦卡锡的网站 www-formal.stanford.edu/jmc/。

[5] 2001 年 3 月 8 日，在加州莫菲特场计算机博物馆历史中心关于“人工智能的起源”的演讲录音，70 分钟。

[6] *Mind*，59 卷第 236 期（1950 年 10 月）“计算机器与智能”。

[7] 麦卡锡 2001 年 3 月 8 日的演讲录音。

[8] 2001 年 5 月 15 日，访谈。

[9] 出自 2000 年 12 月 6 日，麦卡锡访谈。

[10] 1989 年 11 月 1 日，亚瑟·诺伯格对明斯基的访谈，作为查尔斯·巴贝奇研究所口述历史项目的一部分，第 5 页。

[11] 赫伯特·西蒙和艾伦·纽威尔，Heuristic Problem Solving: The Next Advance in Operations Research，*Operation Research*，1958 年 1-2 月，第 1-10 页。

[12] 2001 年 5 月 16 日，访谈。

[13] 2000 年 12 月 6 日，访谈。

[14] 2001 年 3 月 8 日，麦卡锡讲座磁带。

[15] T. A. 怀斯，IBM's $5,000,000,000 Gamble，《财富》，1966 年 9 月，第 119 页。

[16] 珍·萨梅特，The Early History of Cobol，*History of Programming Languages*，Richard L. Wexelblat 编辑。纽约 Academic 出版社 1981 年出版。该论文中包含 Cobol 诞生的大部分纪年表，第 199-242 页。

[17] 若无其他注释，则全部引自珍·萨梅特，大部分背景及作品的描述源自 2000 年 9 月 19 日在她位于马里兰公寓的访谈。

[18] 出自 The Early History of Cobol，第 202 页。

[19] 2000 年 12 月 1 日，访谈。

[20] 罗伯特·贝莫，A View of the History of Cobol，*Honeywell Computer Journal*，第 3 期，第 5 卷（1971 年），第 132 页。

[21] 2001 年 2 月 6 日，访谈。

[22] 在劳伦斯·朱克曼的一篇文章中，霍普对 bug 的故事坚持解释说：“如果词源中有 bug 一词，你将永远无法将它剔除。”2000 年 4 月 22 日，《纽约时报》B 区第 11 页。

[23] 2001 年 3 月 7 日，访谈。

[24] 2001 年 3 月 3 日，访谈。赛尔的访谈对秃头峰会议的描述，细节来自 *IBM's Early Computers*，第 368-371 页。

[25] 引自 T. A. 怀斯，The Rocky Road to the Marketplace，《财富》，1966 年 10 月，第 139 页。

[26] 瓦茨・汉弗雷，Reflections on a Software Life，为卡耐基梅隆大学软件工程学院所写的论文，第 10 页。

[27] 若无其他备注，则全部引自汉弗雷，大部分背景及作品的描述源自 2001 年 5 月 5 日的电话采访。

[28] 2001 年 6 月 26 日，访谈。

[29] 小弗雷德里克・布鲁克斯，*The Mythical Man Month*（1975；1995 再版及改版）。Addison Wesley 出版，第 55 页。

[30] 2000 年 9 月 14 日，访谈。

[31] 坎贝尔-凯利和埃斯普瑞，*Computer*，第 199 页。

[32] 2000 年 9 月 14 日，访谈。

[33] Thoughts on Software Engineering，发表于第十一届国际软件工程大会，1989 年 5 月 15-18 日，匹兹堡，第 97 页。

[34] 1993 年，弗雷德・布鲁克斯，Language Design as Design，该文于 1993 年 4 月 20~23 日会议中提出，同时收录于记录会议进程的书中，*History of Programming Languages II*，小托马斯・伯金和理查德・吉布森编著，ACM Press/Addison-Wesley，1996，第 2-23 页。

第 4 章

[1] 全部引自汤普森，若无其他说明，均引自 2000 年 9 月 22 日在贝尔实验室的访谈以及在他离开贝尔实验室加入加州的创业公司 Entrisphere 之后 2001 年 2 月 7 日的邮件采访。同样，大部分关于其背景及作品的描述均源自这些访谈。背景和细节见普林斯顿大学历史学家迈克尔・S. 马霍尼完成的 Unix 口述历史项目中的记述。

[2] 摘自伯克利自由言论运动的丰富素材，包括 The Daily Californian 网站（www.fsm-a.org）上的数篇文章。

[3] 摘自 2000 年年底，在贝尔实验室举办的汤普森提前退休的聚会上，麦克罗伊写给汤普森的一封信。

[4] 摘自汤普森在贝尔实验室的网站，Cm.bell-labs.com/cm/cs/who/ken/。

[5] 摘自 Reflections on Trusting Trust，*Communication of the ACM*，第 27 卷，第 8 期（1984

年 8 月)，第 761-763 页。

[6] 除非另有注释，全部引自里奇，且大部分对其背景和工作的描述均来自 2000 年 9 月 22 日和 2001 年 1 月 29 日在贝尔实验室的两次访谈，以及 2001 年 2 月 7 日和 2001 年 2 月 15 日的两封电子邮件访谈。

[7] 关于麻省理工学院分时项目的描述摘自对汤普森、里奇以及其他人的访谈。大量关于麻省理工学院项目文化的详细论述来自克鲁兹所著的 *A History of Modern Computing*、坎贝尔–凯利和埃斯普瑞所著的 *Computer*，以及利维所著的 *Hackers*。

[8] 所有科尼汉的引述均摘自 2001 年 2 月 6 日的访谈。

[9] 丹尼斯・里奇和肯・汤普森所著的论文，“The Unix Time-Sharing System”，发表于 1973 年 10 月 15-17 日，在纽约约克镇高地小托马斯・沃森研究中心关于操作系统原则的研讨会。

[10] 道格・麦克罗伊，贝尔实验室的科学家。他关于推动连接程序概念的描述就像数据流，若无其他说明，所有引述摘自 2001 年 2 月 7 日的邮件访谈。

[11] 1964 年 10 月 11 日麦克罗伊所写的备忘录，曾被丹尼斯・里奇粘贴于他贝尔实验室的网站上，cm.bell-labs.com/cm/cs/who/dmr/mdmpipe.html。

[12] 该评论以及后面那个来自乔治・康乐瑞斯的评论引自 2001 年 2 月 16 日的邮件访谈。

[13] 肯尼斯・L. 汤普森所著 *US3568156: Text Matching Algorithm*，归档于 1967 年 8 月 9 日；发表于 1971 年 3 月 2 日。

[14] 明确的 C 语言的发展描述及其前身 BCPL 和 B 语言是丹尼斯・里奇创建的，The Development of the C Language，该论文收录于 *History of Programming Language II*，作者小托马斯・J. 伯金和理查德・G. 吉布森等，纽约：ACM /Addison-Wesley 出版社，1996 年。

第 5 章

[1] 若无其他说明，库尔兹的引述以及关于其背景的一些描述均摘自 2001 年 5 月 29 日和 2001 年 6 月 35 日的两次邮件访谈。

[2] 托马斯・E. 库尔兹的论文“BASIC”于 1978 年 6 月 1-3 日的会议上发表，收录于 *History of Programming Languag*，作者理查德・L. 维克赛尔布莱特等。纽约：1981 年学报，第 516 页。大部分关于 BASIC 语言的发展、背景以及特点的描述摘自库尔兹的论文。

[3] 源于库珀的引述，全部出自 2000 年 12 月 5 日在帕洛阿尔托的访谈以及 2001 年 5 月 29 日和 2001 年 6 月 15 日的邮件访谈。对他的背景及其 Ruby 程序在 Visual Basic 发展过程中所起作用的描述，也都来自上述访谈。

[4] 引自口述历史，Inside Out: Microsoft-In Our Own Words。

[5] 库尔兹的论文“BASIC”，第 518 页。

[6] 有关其前往哥伦比亚及其旅程目的的描述出自 2001 年 1 月 16 日的访谈，以及约瑟夫・F. 特劳布在哥伦比亚大学常规教育研讨会上的论文“What Will Be the Intellectual Impact of Computers？”，见 Proceedings: Educational Frontiers and the Cultural Tradition in the Contemporary University，vol. 11（1982-1983）。

[7] 2001 年 2 月 6 日，访谈。

[8] 有关该谈话的报道以及约翰・麦卡锡关于达特茅斯应该尝试分时系统的提议摘自约翰・G. 凯默尼和托马斯・E. 库尔兹的 *Back to BASIC: The History, Corruption and Future of the Language*（Reading, Mass：Addison-Wesley，1985），第 5-6 页。

[9] 可以找到的最详细的关于凯默尼背景的报道出自斯莱特所著的 *Portraits in Silicon*。

[10] 1968 年的文章 Dartmouth Time-Sharing，作者约翰・凯默尼和托马斯・库尔兹，《科学》杂志第 162 册，第 3850 期（1968 年 10 月 11 日），第 226 页。

[11] 丹尼斯・艾利森，讲师。全部源于艾利森的引述、大部分关于微型 BASIC 同时期的描述出自 2000 年 12 月 5 日在帕洛阿尔托的访谈。关于微型 BASIC 及当时计算机文化的很多描述来自弗雷伯格和司外恩所著的 *Fire in the Valley* 以及利维所著的 *Hackers*。

[12] 引自口述历史，Inside Out: Microsoft-In Our Own Words。

[13] 关于盖茨早期及创建微软的最详细的描述记录于梅尼斯和安德鲁斯所著的 *Gates*。

[14] 见 *Programmers at Work*，作者苏珊拉莫斯等，（华盛顿雷蒙德：微软出版社是，1986 年），第 76 页。

[15] 引自盖茨，下一段引述以及他关于 Visual Basic 的评论出自 2001 年 6 月 22 日的邮件访谈。

[16] 个人计算机革命（包括微软最初的 BASIC）归功于在数字设备公司的微型计算机所做的工作，对此的详细描述见克鲁兹所著的 *A history of Modern Computing*。

[17] 引自口述历史 Inside Out: Microsoft-In Our Own Words。

[18] 所有对尤班克斯的引述以及对其背景的描述摘自 2001 年 5 月 30 日的访谈。

[19] 部分关于 Visual Basic 背景的内容来自 2000 年 10 月 13 日对微软 Visual Basic 和 Visual Studio.Net 部门的经理罗伯・科普兰、戴夫・门德兰和克里斯・迪亚斯的访谈。

[20] 2001 年 6 月 11 日邮件访谈，重温他于 1998 年所做的评论，当时微软涉嫌触犯反垄断法的案件仍在审理过程中。

[21] 2001 年 6 月 11 日访谈。

[22] 2000 年 7 月 12 日，在佛罗里达州奥兰多微软专业软件开发者大会上的讲话。

第 6 章

[1] 坎贝尔–凯利和埃斯普瑞所著的 *Computer*，第 169 页。

[2] 2001 年 2 月 28 日，访谈。

[3] C 语言的起源和命名来自里奇的“C 语言开发”，及本贾尼・斯特劳斯特卢普所著的 *The Design and Evolution of C++*（波士顿：Addison-Wesley/Pearson Education，1994 年出版），第 64 页。

[4] 牛津大学计算机实验室网站上引用斯特雷奇的话，用于阐述本实验室的理念。斯特雷奇卒于 1975 年，离开剑桥之后他一直在牛津从事教学工作。网址为 web.comlab.ox.ac.uk/oucl/about/philosophy.html。

[5] 全部引自斯特劳斯特卢普，若非另外注释，大部分关于其背景和作品的描述源自 2001 年 1 月 31 日在新泽西 AT&T 实验室的访谈，以及 2001 年 2 月 7 日和 2 月 28 日的两次邮件访谈。

[6] 有关斯特劳斯特卢普理念的讨论在 *The Design and Evolution of C++*第 23-25 页中有详细描述。

[7] 关于 Algol、其背景和发展的描述来源广泛。其中两个最有用的是尼古拉斯・沃斯的“Recollections About the Development of Pascal”，美国计算机学会程序设计语言专业组通讯（ACM SIGPLAN Notices），第 28 卷，第 3 期（1993 年 3 月)，和莫里斯・威尔克斯的 *Computing Perspectives*（旧金山： Morgan Kaufmann 出版社， 1995 年）。

[8] 1973 年，美国计算机学会在波士顿召开主题为“编程语言原理”的研讨会上，安东尼・霍尔在其提交的讲稿“Hints on Programming Language Design”中首次提出。最初发表于“技术现状报告”，第 20 卷，第 505-534 页，题目为 Computer Systems Reliability，作者 C. Bunyan 等，Pergamon Infotech 出版社，1974 年。

[9] 威尔克斯的 *Computing Perspectives*，第 95 页。

[10] 尼古拉斯・沃斯，瑞士计算机科学家。他的描述引自“Recollections About the Development of Pascal”。

[11] 2001 年 2 月 27 日，访谈。

[12] 开发 Simula 是为了学习。关于 Simula 的发展背景出自一些印刷资料。霍尔姆维奇（Holmevik）和简・鲁尼（Jan Rune）的文章“The Hostory of Simula”（1995 年，挪威奥斯陆：Institute for Studies in Research and Higher Education 出版社）。该文的早期版本刊登于 *IEEE Annals of the History of Computing*，第 16 卷（4）（1994），第 25-37 页。当前的版本发表于詹姆斯・高斯林的网站：Java.sun.com/people/jag/Simula History.html。其他有关 Simula 的背景资料及克利斯登・奈加特的评论摘自发表于其网站的文章，包

括："How Object Oriented Programming Started"，网址：www.ifi.uio.no/~kristen。及克利斯登・奈加特所著的"Basic Concepts in Object Oriented Programming"，美国计算机学会程序设计语言专业组通讯（ACM SIGPLAN Notices）第 21 卷，第 10 期（1986 年 10 月）。

[13] 见霍尔姆维奇的文章"The History of Simula"，引用奈加特 1992 年的谈话。

[14] 2001 年 3 月 2 日，邮件访谈。

[15] 2001 年 2 月 23 日，邮件访谈。

[16] 2001 年 2 月 26 日，访谈。

第 7 章

[1] 全部引述西蒙尼的话；关于其家庭和作品背景的描述出自分别于 2000 年 10 月 10 日、11 日和 13 日在西蒙尼西雅图家中的访谈，以及 2001 年 3 月 14 日、16 日和 30 日的邮件访谈。在拜访西蒙尼之前，我阅读了拉莫斯（Lammers）的作品 *Programmers at Work*，其中有对西蒙尼的访谈。

[2] 2001 年 3 月 14 日，访谈。

[3] 股票上市时，由《纽约时报》的帕特里克・麦吉汉（Patrick McGeehan）引用投资银行消息人士的话报道，但这一事实并未在《时代周刊》上刊登。

[4] 2001 年 1 月 5 日，访谈。

[5] 关于西蒙尼调试服的描述和萨克尔的引述出自 2001 年 3 月 29 日的邮件访谈。

[6] 对那个时期关于施乐公司帕洛阿尔托研究中心的具体描述参见迈克尔・希尔兹克（Michael Hiltzik）所著的 *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*（纽约：Harper Business 出版社，1999 年）。

[7] 有关 Bravo 发展的描述出自前面提到的对西蒙尼和兰普森的访谈，以及兰普森的论文"Personal Distributed Computing: The Alto and Ethernet Software"，刊登于 *A History of Personal Workstations*（纽约：ACM Press，1988 年），第 293-335 页。

[8] 关于他们对于文档编辑器如何工作的研究，在希尔兹克所著的 *Dealers of Lightning* 中有所描述。

[9] 该引述以及后面盖茨对西蒙尼的描述摘自 2001 年 6 月 22 日的邮件访谈。

[10] 关于微软从西雅图电脑产品公司收购其最初操作系统的描述，见盖茨所著的 *Manes and Andrews*。

[11] 道金斯的评论摘自 2001 年 3 月 22 日的邮件访谈。

[12] 对于国际化编程的描述来自前面提到的对西蒙尼的访谈，以及他的两篇论文。1996 年 6 月 4 日在 IFIP WG 2.1 会议上提交的“Intentional Programming-Innovation in the Legacy Age”和“The Death of Computer Languages, The Birth of Intentional Programming”，Technical Report, MSR-TR-95-52，1995 年 9 月。

第 8 章

[1] 赫兹菲尔德的引述，大部分背景及其 Macintosh 工作的描述摘自 2000 年 12 月 7 日在帕洛阿尔托的访谈，以及 2001 年 4 月 2 日、3 日、11 日和 5 月 30 日的 4 封邮件访谈。

[2] 若无其他注释，其作品及引述引自其论文 Man-Computer Symbiosis，*IRE Transactions on Human Factors in Electronics*，1960 年 3 月：4-11，同时出版于高德伯格编著的 *A History of Personal Workstations*，第 131-140 页。有关利克里德职业的简介见坎贝尔–凯利和埃斯普瑞所著的 *Computer*，第 212-214 页。

[3] 出自 1988 年 10 月 28 日威廉・埃斯普瑞和亚瑟・诺伯格对利克里德访谈的抄本，查尔斯・巴贝奇研究所口述历史项目，第 28 页。

[4] Augmenting Human Intellect: A Conceptual Framework（增强人类智力：一个概念框架）汇总报告，斯坦福研究院，1962 年 10 月。在道格・恩格尔巴特的 Bootstrap 研究所网站上亦可找到，网址为 www.bootstrap.org。

[5] 恩格尔巴特在他的论文 The Augmented Knowledge Workshop 中对他的作品和职业的描述，收录于高德伯格编著的 *A History of Personal Workstations* 第 187-232 页，同时收录了问答部分，见第 233-236 页。恩格尔巴特的其余引述均来自上述论文。

[6] 克鲁兹所著的 *A History of Modern Computing* 对施乐公司帕洛阿尔托研究中心的工作，以及与利克里德和恩格尔巴特所做研究的债务做出了简要总结，见第 257-263 页。希尔兹克所著的 *Dealers of Lightning*，是关于 20 世纪 70 年代施乐公司帕洛阿尔托研究中心最详细的描述。

[7] 大部分关于凯伊计算机背景及其在施乐公司帕洛阿尔托研究中心工作的描述，摘自其论文 The Early History of Smalltalk，美国计算机学会程序设计语言专业组通讯第 28 卷，第 3 期（1993 年 3 月），第 1-52 页。该论文于 1993 年 4 月 20~23 日的会议上提交，收录于记录会议，*History of Programming Languages II*，小托马斯・伯金和理查德・吉布森编著，纽约：ACM Press/Addison-Wesley，1996 年。

[8] 凯伊对 Smalltalk 的叙述及其后续的引用来自 2001 年 4 月 2 日和 4 日的邮件访谈。

[9] 2001 年 1 月 5 日，访谈。

[10] 2001 年 3 月 14 日，访谈。

[11] 拉斯金曾指出他本人并不赞同 Macintosh 历史中给予史蒂夫·乔布斯过高的赞誉。在他的文章 Holes in History 中提到过这些，见 ACM 出版的 *interactions*，1994 年 7 月，第 11-16 页。有关 Macintosh 项目最详尽的描述见史蒂芬·利维所著的 *Insanely Great: The Life and Times of Macintosh, the Computer that Changed Everything*（纽约：Penguin 出版社，1994 年）。

[12] 有关特里布尔与比尔·阿特金森友谊的描述，以及特里布尔的引述，出自 2000 年 9 月 26 日在帕洛阿尔托的访谈。

[13] 卡普斯的全部引述及其对于 Macintosh 所做努力的描述，均出自 2000 年 10 月 10 日在西雅图的访谈、2000 年 12 月 8 日在加州 San Carlos 的访谈以及 2001 年 4 月 2 日的邮件访谈。

第 9 章

[1] 贝什等所著的 IBM'sEarlyComputers，第 474-480 页。

[2] 若无其他注释，有关钱伯林的引述以及对其作品和背景的描述，均出自 2000 年 12 月 8 日在圣何塞的访谈，以及 2001 年 4 月 17 日和 23 日的两次邮件访谈。

[3] 摘自钱伯林于 2000 年 5 月在加利福尼亚大学戴维斯分校计算机科学系座谈会上的演讲，题为 ABriefHistoryofData。

[4] 查尔斯·W. 巴赫曼的论文 TheProgrammerasNavigator，见 ACM 通讯第 16 卷，第 11 期（1973 年），第 635-658 页。

[5] 有关科德参观约克镇高地以及钱伯林对此次参观的相关评论引自保罗麦·克琼斯编辑的 The1995SQLReunion:People,Projects,andPolitics。科学研究委员会技术说明（SRCTechnicalNote）1997-018。（该论文是 1995 年 5 月 28~30 日在加州太平洋丛林市召开会议的记录，由 DigitalEquipment’sSystemsResearchCenter 出版）。

[6] E.F.科德所著的 *ARelationalModelofDataforLargeSharedDataBanks*，见通讯 ACM 第 13 卷，第 6 期（1970 年），第 377-387 页。

[7] 所有格雷引述以及大部分对其背景和作品的描述，均出自 2000 年 9 月 28 日在旧金山的访谈和 2000 年 4 月 14 日的邮件访谈。

[8] 人们提及研究与创新时常常引述这句话。但是当在使用谷歌快速搜索时，我注意到按照凯伊的引述，智商的数值范围是从 50 到 100，而通常以“context”和“perspective”代替“pointofview”，因此我写邮件给阿兰·凯伊，询问他最初的引述是什么，他于 2001 年 4 月 19 日给了回复。

[9] 即唐·和雷·博伊斯的文章 SEQUEL:AStructuredEnglishQueryLanguage，见 ACM 数据管理专业委员会于 1974 年 5 月在密歇根州安阿伯市召开的会议论文集，第 249-264 页，会议主题为“数据/描述、存取和控制”。

[10] 布莱克林的引述和大部分关于其作品以及背景的描述，均出自 2001 年 4 月 17 日的电话访谈。在访谈之前，为了大致了解布莱克林，我阅读了斯莱特所著的 PortraitsinSilicon。

[11] 弗兰克斯顿的引述和大部分关于其作品及背景的描述，均出自 2001 年 1 月 13 日在其位于马萨诸塞州堪布里奇（Cambridge，Massachusetts）家中的访谈，以及 2001 年 4 月 13 日的邮件访谈。

[12] 2001 年 1 月 5 日，访谈。

[13] 罗森在 1979 年的一份报告的扫描件，VisiCalc:BreakingthePersonalComputerSoftware-Bottleneck，参见丹・布莱克林的网站 www.bricklin.com。网站上同时有布莱克林关于 VisiCalc 产生过程的描述，以及那个时期的一些照片。

[14] 布拉斯根的引述，见 The1995SQLReunion 的抄本，第 34 页。

[15] 2000 年 9 月 26 日，访谈。

[16] 若无其他注释，伯纳斯–李的引述及一些背景的描述，均引自 1995 年 12 月 13 日和 1999 年 9 月 16 日的两次访谈。大部分关于其早期工作经历，以及其作品和思想演变的详细描述，参见他在 1999 年与马克・菲谢蒂合著的 *WeavingtheWeb：TheOriginalDesignandUltimateDestinyoftheWorldWideWebbyItsCreator*（旧金山：HarperSanFrancisco 出版社，1999）一书。

[17] 2000 年 9 月 26 日，访谈。

第 10 章

[1] 高斯林的全部引述以及大部分关于其背景、作品的描述，均引自 2000 年 9 月 26 日和 28 日在加利福尼亚山景城的访谈、2001 年 1 月 12 日的电话访谈和 2000 年 6 月 29 日的邮件访谈。1995 年 2 月，他还撰写了太阳微系统公司“白皮书”，其中对 Java 语言有很好的叙述，题为 Java: An Overview。

[2] 斯宾塞・瑞斯所著的 Power to the People 中对盖奇的背景有很好的介绍，见《连线》杂志，1996 年 12 月。

[3] 2001 年 1 月 12 日，访谈。

[4] 2001 年 1 月 11 日，访谈。

[5] 2001 年 1 月 16 日，访谈。

[6] 2001 年 1 月 11，访谈。

[7] 谢里登的引述及其对 Java 项目早期氛围的描述，引自 2000 年 11 月 15 日的访谈，以及 2001 年 1 月 9 日和 6 月 29 日的邮件访谈。

[8] "绿色大门背后的秘密：关于消费电子产品机遇的深入思考"。1991 年 8 月 23 日，由 Green 团队成员提出（高斯林给过我一份复印件）。

[9] 上述细节及其他内容首次出版于的《连线》杂志（1995 年 12 月）中的一篇文章 The Java Saga，作者戴维·班克。在对高斯林进行访谈之前，我曾读过戴维·班克的文章和其他一些材料。

[10] 乔伊的引述以及对其在 Java 方面成就的叙述，均引自 2001 年 1 月 10 日在纽约的访谈，以及 2000 年 10 月 9 日和 2001 年 1 月 15 日的邮件访谈。

[11] 施密特的引述以及对其 Java 商业开发战略的叙述，均引自 2001 年 1 月 15 日的访谈。

[12] 2001 年 1 月 26 日，访谈。

[13] 斯蒂尔的引述以及其对 Java 会议的描述，均引自 2001 年 1 月 19 日的访谈。

[14] 安德森的引述出自 2001 年 1 月 23 日的访谈。

[15] 2000 年 10 月 11 日，访谈。

[16] 2000 年 7 月 10 日，访谈。

第 11 章

[1] 贝伦多夫的全部引述以及大部分关于其背景和作品的描述，均引自 2000 年 8 月 8 日和 9 月 26 日在旧金山的访谈，以及 2000 年 10 月 3 日、2001 年 5 月 29 日和 31 日的邮件访谈。

[2] 特布什的全部引述及其对阿帕奇项目的描述，均出自 2000 年 12 月 5 日的访谈，以及 2001 年 5 月 31 的邮件访谈。

[3] 摘自 The Netcraft Web Server Survey，结果产生于 2001 年 4 月，并于 2001 年 5 月报告。Netcraft 是对接入互联网的计算机所使用的 Web 服务器软件进行调查的公司。在其网站上有调查的内容，网址为 www.netcraft.com。

[4] 埃里克·S. 雷蒙德的文章 The Magic Cauldron，见其著作 *The Cathedral and the Bazaar*（加利福尼亚州塞巴斯托波：O'Reilly & Associates 出版社，1999 年），第 137-194 页。

[5] 斯托曼的所有引述以及大部分关于其背景和作品的描述，均摘自 2000 年 12 月 23 日在纽约的访谈。利维的 *Hackers* 一书详尽记录了斯托曼在麻省理工学院人工智能实验室的日子以及自由软件运动的开端。

[6] 林纳斯·托沃兹和戴维·戴蒙德所著的 *Just for Fun: The Story of an Accidental Revolutionary*（纽约：Harper Business 出版社，2001）第 54 页（校样）。该书对其背景和 Linux 的产生与发展进行了最详细的描述。

[7] 雷蒙德所著的 *The Cathedral and the Bazaar*，第 63 页。

[8] 若无其他注释，托沃兹的所有引述均出自 2001 年 3 月 6 日的邮件访谈。

[9] 林纳斯・托沃兹的文章 The Linux Edge，见 *Open Sources: Voices from the Open Source Revolution*，克里斯迪・博纳、山姆・奥克曼和马克・斯通编著，（加利福尼亚州塞巴斯托波：O'Reilly & Associates 出版社，1999），第 107 页。

[10] 瓦拉达斯基・博格的大部分引述及其对教育与编程经验的描述，引自 2001 年 3 月 28 日在纽约的访谈，以及 2000 年 7 月 3 日、8 月 2 日和 2001 年 5 月 30 日的邮件访谈。有关其早期个人背景的大部分描述，引自我写的瓦拉达斯基・博格传略：Well, Somebody's Got to Reinvent the IBM Mainframe，1993 年 9 月 12 日《纽约时报》，周日商业版第 8 页。

[11] 有关 IBM 对 Linux 的研究，以及对相关报告和管理层电子邮件的引用，来自我为《纽约时报》撰写的两篇专题文章。2000 年 1 月 10 日的 IBM to Use Linux System in Software for Internet，第 3 节，第 1 页，和 2000 年 3 月 20 日的 A Mainstream Giant Goes Counterculture: IBM's Embrace of Linux a Bet That It Is the Software of the Future，第 3 节，第 1 页。

[12] 摘自报告 Free Software/Open Source: Information Society Opportunities for Europe? 该报告由自由软件工作团队完成，该团队是应欧盟委员会信息社会执行总署（Information Society Directorate General of the European Commission）的要求而组建的。2000 年 4 月 1.2 版本。该报告网址为 eu.conecta.ita。

[13] 2001 年 5 月 30 日，访谈。

[14] 乔伊的评论来自 2000 年 8 月 6 日和 9 月 26 日的访谈。

[15] 2001 年 1 月 31 日，访谈。

参考文献

书

Bashe, Charles J., Lyle R. Johnson, John H. Palmer, and Emerson W. Pugh. *IBM's Early Computers*. Cambridge, Mass.: MIT Press, 1986

Berners-Lee, Tim with Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Creator*. San Francisco: HarperSanFrancisco, 1999.

Bernstein, Jeremy. *The Analytical Engine: Computers – Past, Present, and Future*. New York: Random House, 1963.

Brooks, Frederick P., Jr. *The Mythical Man-Month*. Reading, Mass.: Addison-Wesley Publishing, 1975 (reprinted and updated 1995)..

Campbell-Kelly, Martin, and William Aspray. *Computer: A History of the Information Machine*. New York: Basic Books, 1996.

Ceruzzi, Paul E. *A History of Modern Computing*. Cambridge, Mass.: MIT Press, 1998.

Cusumano, Michael A. and Richard W. Selby. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology*, *Shapes Markets and Manages People*. New York: Free Press, 1995.

Dibona, Chris, Sam Ockman, and Mark Stone, eds. *Open Sources: Voices from the Open Source Revolution*. Sebastopol, Calif.: O'Reilly & Associates, 1999.

Freiberger, Paul, and Michael Swaine. *Fire in the Valley: The Making of the Personal Computer*. New York: McGraw-Hill, 1984 (updated edition 2000).

Goldberg, Adele, ed. *A History of Personal Workstations*. New York: ACM Press, 1988.

Hillis, W. Daniel. *The Pattern on the Stone: The Simple Ideas that Make Computers Work*. New York: Basic Books, 1998.

Hiltzik, Michael. *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age*. New York: Harper Business, 1999.

Jersome, Kelli, and Marlee Anderson, eds. *Inside Out: Microsoft – In Our Own Words*. New York: Warner Books, 2000.

Kemeny, John G. and Thomas E. Kurtz. *Back to BASIC: The History, Corruption and Future of the Language*. Reading, Mass: Addison-Wesley, 1985.

Knuth, Donald E. *The Art of Computer Programming: Volume 1 Fundamental Algorithms*. Reading, Mass.: Addison-Wesley, 1997 (third edition).

Lammers, Susan, ed. *Programmers at Work*. Redmond, Wash.: Microsoft Press, 1986.

Levy, Steven. *Hackers: Heroes of the Computer Revolution*. New York: Doubleday, 1984.

Levy, Steven. *Insanely Great: The Life and Times of Macintosh, the Computer that Changed Everything*. New York: Penguin, 1994.

Manes, Stephen, and Paul Andrews. *Gates: How Microsoft's Mogul Reinvented an Industry – and Made Himself the Richest Man in America*. New York: Doubleday, 1993.

McCartney, Scott. *ENIAC: The Triumphs and Tragedies of the World's First Computer*. New York: Walker & Company, 1999.

Pugh, Emerson, and Lyle R. Johnson and John H. Palmer. *IBM's 360 and Early 370 Systems*. Cambridge, Mass.: MIT Press, 1991.

Raymond, Eric S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, Calif.: O'Reilly & Associates, 1999.

Slater, Robert. *Portraits in Silicon*. Cambridge, Mass.: MIT Press, 1987.

Stroustrup, Bjarne. *The Design and Evolution of* C++. Boston: Addison-Wesley/Pearson Education, 1994.

Torvalds, Linus, and David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. New York: Harper Business, 2001.

Wilkes, Maurice V., and David J. Wheeler and Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer: With Special Reference to the EDSAC and the Use of a Library of Subroutines*. Reading, Mass: Addison-Wesley, 1951.

Wilkes, Maurice. *Memoirs of a Computer Pioneer*. Cambridge, Mass.: MIT Press, 1985.

Wilkes, Maurice V. *Computing Perspectives*. San Francisco: Morgan Kaufmann, 1995.

Weizenbaum, Joseph. *Computer Power and Human Reason: From Judgment to Calculation*. W. H. Freeman & Company, 1976.

文章及论文

Bachman, Charles W. “The Programmer as Navigator.” *Communications of the ACM*, vol. 16, no. 11, (1973). pp. 635–58.

Backus, John. *The History of Fortran I, II and III*. A paper originally published in the *Annals of the History of Computing* vol. 1, no. 1 (July 1979). It was reprinted in the *IEEE Annals of the History of Computing* vol. 20, no. 4 (1998), pp. 68–78.

Bank, David. “The Java Saga.” Wired, December 1995.

Brooks, Frederick P. Jr. “Language Design as Design.” The paper was presented at a conference on April 20–23, 1993, and included in a book on the proceedings, *History of Programming languages II*, Thomas J. Bergin, Jr. and Richard G. Gibson, eds. New York: ACM Press/Addison-Wesley, 1996, pp. 3–23.

Bush, Vannewar. “As We May Think.” *The Atlantic Monthly*, July 1945. pp. 101–8.

Codd, E. F. “A Relational Model of Data for Large Shared Data Banks.” *Communications of the ACM* vol. 13, no. 6 (June 1970), pp. 377–87.

Chamberlin D. D. and R. F. Boyce. “SEQUEL: A Structured English Query Language.” *Proceedings of the ACM SIGMOD on Data Description, Access and Control*. Ann Arbor, Mich. (May 1974), pp. 249–64.

Dijkstra, Edsger W. “Go To Statement Considered Harmful.” *Communications of the ACM* vol. 11, no. 3 (March 1968), pp. 147–48.

“Preliminary Report: Specifications for The IBM Mathematical FOR mula TRANslation System.” Programming Research Group, Applied Science Division, International Business Machines Corporation, November 11, 1954.

Holmevik, Jan Rune. “The History of Simula.” Oslo, Norway: Institute for Studies in Research and Higher Education, 1995. An earlier version of the article was published in *IEEE Annals of the History of Computing* 16 (4) (1994), pp. 25–37. A copy of the current version was posted on James Gosling’s Web site, at java.sun.com/people/jag/SimulaHistory.html.

Kay, Alan. “The Early History of Smalltalk.” *ACM Sigplan Notices* vol. 28, no. 3 (March 1993), pp. 1–52. The paper was presented at a conference on April 20–23, 1993, and is included in the book of the proceedings, *History of Programming Languages II*, Thomas J. Bergin Jr. and Richard G. Gibson, eds. New York: ACM Press/Addison-Wesley, 1996.

Kemeny, John G. and Thomas E. Kurtz. “Dartmouth Time-Sharing.” *Science* 1968. vol. 162, no. 3850 (October 11, 1968).

Lee, J. A. N. “Programming Languages, Past, Present and Future.” *IEEE looking.forward*, Student Newsletter, Fall 1996.

Lee, J. A. N., and H. S. Tropp, eds. “Special Issue: Fortran’s Twenty-Fifth Anniversary.” Vol. 6, No. 1, January 1984.

Licklider, J. C. R. “Man–Computer Symbiosis.” *IRE Transactions on Human Factors in Electronics*, March 1960, pp. 4–11; also reprinted in Goldberg ed. *A History of Personal Workstations*, pp. 131–140.

Ritchie, D. M. and K. Thompson. "The Unix Time-Sharing System." Paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, October 15–17, 1973.

Ritchie, Dennis M. "The Development of the C Language." The paper was presented at a conference on April 20–23, 1993, and included in a book of the proceedings, *History of Programming Languages II*, Thomas J. Bergin, Jr. and Richard G. Gibson, eds. NewYork: ACM Press/Addison-Wesley, 1996.

Sammet, Jean E. "The Early History of Cobol." A paper presented at a conference on June 1–3, 1978, and included in a book of the proceedings, *History of Programming Languages*, Richard L. Wexelblat, ed. New York: Academic Press, 1981.

Thompson, Kenneth. "Reflections on Trusting Trust." *Communication of the ACM* vol. 27, no. 8 (August 1984), pp. 761–63.

Traub, Jospeph F. "What Will Be the Intellectual Impact of Computers?" *Proceedings: Educational Frontiers and the Cultural Tradition in the Contemporary University.* Vol. 11, 1982–83. From a General Education Seminar, Columbia University.

口述历史

(All from the oral history project of the Charles Babbage Institute at the University of Minnesota.)

Corbato, Fernando J. Interviewed by Arthur L. Norberg, April 18, 1989 and November 14, 1990.

Hopper, Grace. Interviewed by Christopher Evans, ca. 1976.

Licklider, J. C. R. Interviewed by conducted by William Aspray and Arthur Norberg, October 28, 1988.

Minsky, Marvin L. Interviewed by Arthur L. Norberg, November 1, 1989.

Wheeler, David J. Interviewed by William Aspray, May 14, 1987.

录像带

Interviews with members of Fortran team. IBM documentary film, from IBM Archives. Produced by Bright Star Films. 13 minutes. May 26, 1982.

McCarthy, John. Tape of lecture at the Computer Museum History Center, Moffett Field, Calif., on the "origins of artificial intelligence." 70 minutes. March 8, 2001.

Allen, Fran. Tape of lecture at the Computer Museum History Center, Moffett Field, Calif., on the government's Stretch/Harvest computer. 114 minutes. November 8, 2000.

人名索引

D

F

G

J

K

L

M

N

O

Q

S

T

W

X

Y

Z

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks